INTERNATIONAL STANDARD

ISO/IEC 14496-2

Third edition
2004-06-01

Information technology — Coding of audio-visual objects — Part 2: Visual

Technologies de l'information — Codage des objets audiovisuels — Partie 2: Codage visuel

Reference number
ISO/IEC 14496-2:2004(E)

© ISO/IEC 2004

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 2004

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2004 – All rights reserved

Not for Resale

Contents

1 Scope ... 1

2 Normative references ... 1

3 Terms and definitions ... 1

4 Abbreviations and symbols ... 13
4.1 Arithmetic operators ... 13
4.2 Logical operators ... 14
4.3 Relational operators ... 14
4.4 Bitwise operators ... 15
4.5 Conditional operators ... 15
4.6 Assignment ... 15
4.7 Mnemonics ... 15
4.8 Constants ... 15

5 Conventions ... 16
5.1 Method of describing bitstream syntax ... 16
5.2 Definition of functions ... 17
5.3 Reserved, forbidden and marker_bit ... 18
5.4 Arithmetic precision ... 19

6 Visual bitstream syntax and semantics ... 19
6.1 Structure of coded visual data ... 19
6.2 Visual bitstream syntax ... 38
6.3 Visual bitstream semantics ... 135

7 The visual decoding process ... 236
7.1 Video decoding process ... 237
7.2 Higher syntactic structures ... 238
7.3 VOP reconstruction ... 238
7.4 Texture decoding ... 239
7.5 Shape decoding ... 250
7.6 Motion compensation decoding ... 274
7.7 Interlaced video decoding ... 297
7.8 Sprite decoding ... 306
7.9 Generalized scalable decoding ... 313
7.10 Still texture object decoding ... 323
7.11 Mesh object decoding ... 347
7.12 FBA object decoding ... 352
7.13 3D Mesh Object Decoding ... 358
7.14 NEWPRED mode decoding ... 384
7.15 Output of the decoding process ... 385
7.16 Video object decoding for the studio profile ... 385
7.17 The FGS decoding process ... 427

8 Visual-Systems Composition Issues ... 429
8.1 Temporal Scalability Composition ... 429
8.2 Sprite Composition ... 430
8.3 Mesh Object Composition ... 431
8.4 Spatial Scalability composition ... 432

9 Profiles and Levels ... 432
9.1 Visual Object Types ... 432
9.2 Visual Profiles ... 436
9.3 Visual Profiles@Levels ... 437

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A (normative) **Coding transforms** **441**
Annex B (normative) **Variable length codes and arithmetic decoding** **451**
Annex C (normative) **Face and body object decoding tables and definitions** **547**
Annex D (normative) **Video buffering verifier** **580**
Annex E (informative) **Features supported by the algorithm** **589**
Annex F (informative) **Preprocessing and postprocessing** **599**
Annex G (normative) **Profile and level indication and restrictions** **625**

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 14496-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This third edition cancels and replaces the second edition (ISO/IEC 14496-2:2001), which has been technically revised. It also incorporates the Amendments ISO/IEC 14496-2:2001/Amd. 1:2002, ISO/IEC 14496-2:2001/Amd. 2:2002 and ISO/IEC 14496-2:2001/Amd. 3:2003.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects*

— *Part 8: Carriage of ISO/IEC 14496 content over IP networks*

— *Part 9: Reference hardware description*

— *Part 10: Advanced video coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extentions*

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *Part 14: MP4 file format*

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation framework extension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

Purpose

This part of ISO/IEC 14496 was developed in response to the growing need for a coding method that can facilitate access to visual objects in natural and synthetic moving pictures and associated natural or synthetic sound for various applications such as digital storage media, internet, various forms of wired or wireless communication etc. The use of ISO/IEC 14496 means that motion video can be manipulated as a form of computer data and can be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcast channels.

Application

The applications of ISO/IEC 14496 cover, but are not limited to, such areas as listed below:

IMM Internet Multimedia

IVG Interactive Video Games

IPC Interpersonal Communications (videoconferencing, videophone, etc.)

ISM Interactive Storage Media (optical disks, etc.)

MMM Multimedia Mailing

NDB Networked Database Services (via ATM, etc.)

RES Remote Emergency Systems

RVS Remote Video Surveillance

WMM Wireless Multimedia

Multimedia

Profiles and levels

ISO/IEC 14496 is intended to be generic in the sense that it serves a wide range of applications, bitrates, resolutions, qualities and services. Furthermore, it allows a number of modes of coding of both natural and synthetic video in a manner facilitating access to individual objects in images or video, referred to as content based access. Applications should cover, among other things, digital storage media, content based image and video databases, internet video, interpersonal video communications, wireless video etc. In the course of creating ISO/IEC 14496, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and they have been integrated into a single syntax. Hence ISO/IEC 14496 will facilitate the bitstream interchange among different applications.

This part of ISO/IEC 14496 includes one or more complete decoding algorithms as well as a set of decoding tools. Moreover, the various tools of this part of ISO/IEC 14496 as well as that derived from ISO/IEC 13818-2:2000 can be combined to form other decoding algorithms. Considering the practicality of implementing the full syntax of this part of ISO/IEC 14496, however, a limited number of subsets of the syntax are also stipulated by means of “profile” and “level”.

A “profile” is a defined subset of the entire bitstream syntax that is defined by this part of ISO/IEC 14496. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by parameters in the bitstream.

In order to deal with this problem “levels” are defined within each profile. A level is a defined set of constraints imposed on parameters in the bitstream. These constraints may be simple limits on numbers. Alternatively they may take the form of constraints on arithmetic combinations of the parameters.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Object based coding syntax

Video object

A *video object* in a scene is an entity that a user is allowed to access (seek, browse) and manipulate (cut and paste). The instances of video objects at a given time are called *video object planes* (VOPs). The encoding process generates a coded representation of a VOP as well as composition information necessary for display. Further, at the decoder, a user may interact with and modify the composition process as needed.

The full syntax allows coding of rectangular as well as arbitrarily shaped video objects in a scene. Furthermore, the syntax supports both nonscalable coding and scalable coding. Thus it becomes possible to handle normal scalabilities as well as object based scalabilities. The scalability syntax enables the reconstruction of useful video from pieces of a total bitstream. This is achieved by structuring the total bitstream in two or more layers, starting from a standalone base layer and adding a number of enhancement layers. The base layer can be coded using a non-scalable syntax, or in the case of picture based coding, even using a syntax of a different video coding standard.

To ensure the ability to access individual objects, it is necessary to achieve a coded representation of its shape. A natural video object consists of a sequence of 2D representations (at different points in time) referred to here as VOPs. For efficient coding of VOPs, both temporal redundancies as well as spatial redundancies are exploited. Thus a coded representation of a VOP includes representation of its shape, its motion and its texture.

FBA object

The FBA object is a collection of nodes in a scene graph which are animated by the FBA (Face and Body Animation) object bitstream. The FBA object is controlled by two separate bitstreams. The first bitstream, called BIFS, contains instances of Body Definition Parameters (BDPs) in addition to Facial Definition Parameters (FDPs), and the second bitstream, FBA bitstream, contains Body Animation Parameters (BAPs) together with Facial Animation Parameters (FAPs).

A 3D (or 2D) *face object* is a representation of the human face that is structured for portraying the visual manifestations of speech and facial expressions adequate to achieve visual speech intelligibility and the recognition of the mood of the speaker. A face object is animated by a stream of *face animation parameters (FAP)* encoded for low-bandwidth transmission in broadcast (one-to-many) or dedicated interactive (point-to-point) communications. The FAPs manipulate key feature control points in a mesh model of the face to produce animated visemes for the mouth (lips, tongue, teeth), as well as animation of the head and facial features like the eyes. FAPs are quantised with careful consideration for the limited movements of facial features, and then prediction errors are calculated and coded arithmetically. The remote manipulation of a face model in a terminal with FAPs can accomplish lifelike visual scenes of the speaker in real-time without sending pictorial or video details of face imagery every frame.

A simple streaming connection can be made to a decoding terminal that animates a default face model. A more complex session can initialize a custom face in a more capable terminal by downloading *face definition parameters (FDP)* from the encoder. Thus specific background images, facial textures, and head geometry can be portrayed. The composition of specific backgrounds, face 2D/3D meshes, texture attribution of the mesh, etc. is described in ISO/IEC 14496-1:2001. The FAP stream for a given user can be generated at the user's terminal from video/audio, or from text-to-speech. FAPs can be encoded at bitrates up to 2-3kbit/s at necessary speech rates. Optional temporal DCT coding provides further compression efficiency in exchange for delay. Using the facilities of ISO/IEC 14496-1:2001, a composition of the animated face model and synchronized, coded speech audio (low-bitrate speech coder or text-to-speech) can provide an integrated low-bandwidth audio/visual speaker for broadcast applications or interactive conversation.

Limited scalability is supported. Face animation achieves its efficiency by employing very concise motion animation controls in the channel, while relying on a suitably equipped terminal for rendering of moving 2D/3D faces with non-normative models held in local memory. Models stored and updated for rendering in the terminal can be simple or complex. To support speech intelligibility, the normative specification of FAPs intends for their selective or complete use as signaled by the encoder. A masking scheme provides for selective transmission of FAPs according to what parts of the face are naturally active from moment to moment. A further control in the FAP stream allows face animation to be suspended while leaving face features in the terminal in a defined quiescent state for higher overall efficiency during multi-point connections.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A body model is a representation of a virtual human or human-like character that allows portraying body movements adequate to achieve nonverbal communication and general actions. A body model is animated by a stream of *body animation parameters* (BAP) encoded for low-bitrate transmission in broadcast and dedicated interactive communications. The BAPs manipulate independent degrees of freedom in the skeleton model of the body to produce animation of the body parts. The BAPs are quantised considering the joint limitations, and prediction errors are calculated and coded arithmetically. Similar to the face, the remote manipulation of a body model in a terminal with BAPs can accomplish lifelike visual scenes of the body in real-time without sending pictorial and video details of the body every frame.

The BAPs, if correctly interpreted, will produce reasonably similar high level results in terms of body posture and animation on different body models, also without the need to initialize or calibrate the model.The BDP set defines the set of parameters to transform the default body to a customized body optionally with its body surface, body dimensions, and texture.

The *body definition parameters (BDP)* allow the encoder to replace the local model of a more capable terminal. BDP parameters include body geometry, calibration of body parts, degrees of freedom, and optionally deformation information.

The FBA Animation specification is defined in ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496. This clause is intended to facilitate finding various parts of specification. As a rule of thumb, FAP and BAP specification is found in the part 2, and FDP and BDP specification in the part 1. However, this is not a strict rule. For an overview of FAPs/BAPs and their interpretation, read subclauses “6.1.5.2 Facial animation parameter set”, “6.1.5.3 Facial animation parameter units”, “6.1.5.4 Description of a neutral face” as well as the Table C.1. The viseme parameter is documented in subclause “7.12.3 Decoding of the viseme parameter fap 1” and the Table C.5 in Annex C. The expression parameter is documented in subclause “7.12.4 Decoding of the expression parameter fap 2” and the Table C.3. FBA bitstream syntax is found in subclauses “6.2.10 FBA Object”, semantics in “6.3.10 FBA Object”, and subclause “7.12 FBA object decoding” explains in more detail the FAP/BAP decoding process. FAP/BAP masking and interpolation is explained in subclauses “6.3.11.1 FBA Object Plane”, “7.12.1.1 Decoding of FBA”, “7.12.5 FBA masking” . The FIT interpolation scheme is documented in subclause “7.2.5.3.2.4 FIT” of ISO/IEC 14496-1:2001. The FDPs and BDPs and their interpretation are documented in subclause “7.2.5.3.2.6 FDP” of ISO/IEC 14496-1:2001. In particular, the FDP feature points are documented in Figure C-1. Details on body models are documented in Annex C.

Mesh object

A 2D *mesh object* is a representation of a 2D deformable geometric shape, with which synthetic video objects may be created during a composition process at the decoder, by spatially piece-wise warping of existing video object planes or still texture objects. The instances of mesh objects at a given time are called *mesh object planes* (mops). The geometry of mesh object planes is coded losslessly. Temporally and spatially predictive techniques and variable length coding are used to compress 2D mesh geometry. The coded representation of a 2D mesh object includes representation of its geometry and motion.

3D Mesh Object

The 3D Mesh Object is a 3D polygonal model that can be represented as an IndexedFaceSet or Hierarchical 3D Mesh node in BIFS. It is defined by the position of its vertices (geometry), by the association between each face and its sustaining vertices (connectivity), and optionally by colours, normals, and texture coordinates (properties). Properties do not affect the 3D geometry, but influence the way the model is shaded. 3D mesh coding (3DMC) addresses the efficient coding of 3D mesh object. It comprises a basic method and several options. The basic 3DMC method operates on manifold model and features incremental representation of single resolution 3D model. The model may be triangular or polygonal – the latter are triangulated for coding purposes and are fully recovered in the decoder. Options include: (a) support for computational graceful degradation control; (b) support for non-manifold model; (c) support for error resilience; and (d) quality scalability via hierarchical transmission of levels of detail with implicit support for smooth transition between consecutive levels. The compression of application-specific geometry streams (Face Animation Parameters) and generalized animation parameters (BIFS Anim) are currently addressed elsewhere in this part of ISO/IEC 14496.

In 3DMC, the compression of the connectivity of the 3D mesh (e.g. how edges, faces, and vertices relate) is lossless, whereas the compression of the other attributes (such as vertex coordinates, normals, colours, and texture coordinates) may be lossy.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Single Resolution Mode

The incremental representation of a single resolution 3D model is based on the Topological Surgery scheme. For manifold triangular 3D meshes, the Topological Surgery representation decomposes the connectivity of each *connected component* into a *simple polygon* and a *vertex graph*. All the triangular faces of the 3D mesh are connected in the simple polygon forming a *triangle tree*, which is a spanning tree in the dual graph of the 3D mesh. Figure 0-1 shows an example of a triangular 3D mesh, its dual graph, and a triangle tree. The vertex graph identifies which pairs of boundary edges of the simple polygon are associated with each other to reconstruct the connectivity of the 3D mesh. The triangle tree does not fully describe the triangulation of the simple polygon. The missing information is recorded as a *marching edge.*

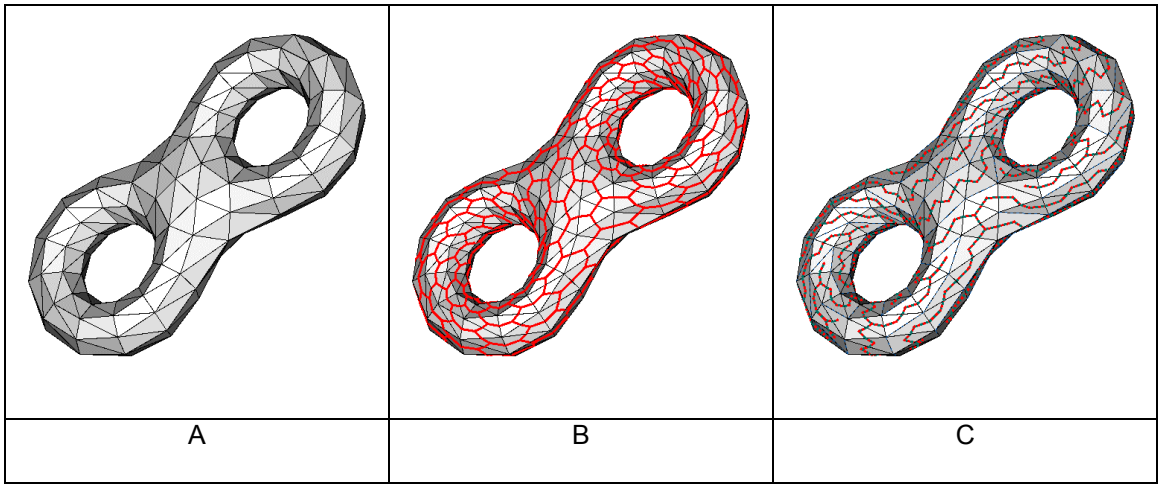


Figure 0-1 — A triangular 3D mesh (A), its dual graph (B), and a triangle tree (C)

For manifold 3D meshes, the connectivity is represented in a similar fashion. The polygonal faces of the 3D mesh are connected in a simple polygon forming a *face tree*. The faces are triangulated, and which edges of the resulting triangular 3D mesh are edges of the original 3D mesh is recorded as a sequence of *polygon_edge* bits. The face tree is also a spanning tree in the dual graph of the 3D mesh, and the vertex graph is always composed of edges of the original 3D mesh.

The vertex coordinates and optional properties of the 3D mesh (normals, colours, and texture coordinates) are quantised, predicted as a function of decoded ancestors with respect to the order of traversal, and the errors are entropy encoded.

Incremental Representation

When a 3D mesh is downloaded over networks with limited bandwidth (e.g. PSTN), it may be desired to begin decoding and rendering the 3D mesh before it has all been received. Moreover, content providers may wish to control such incremental representation to present the most important data first. The basic 3DMC method supports this by interleaving the data such that each triangle may be reconstructed as it is received. Incremental representation is also facilitated by the options of hierarchical transmission for quality scalability and partitioning for error resilience.

Hierarchical Mode

An example of a 3D mesh represented in hierarchical mode is illustrated in Figure 0-2. The hierarchical mode allows the decoder to show progressively better approximations of the model as data are received. The hierarchical 3D mesh decomposition can also be organized in the decoder as layered detail, and view-dependent expansion of this detail can be subsequently accomplished during a viewer's interaction with the 3D model. Downloadable scalability of 3D model allows decoders with widely varied rendering performance to use the content, without necessarily making repeated requests to the encoder for specific versions of the content and without the latencies of updating view-dependent model from the encoder. This quality scalability of 3D meshes is complementary to scaleable still texture and supports bitstream scalability and downloading of quality hierarchies of hybrid imagery to the decoder.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

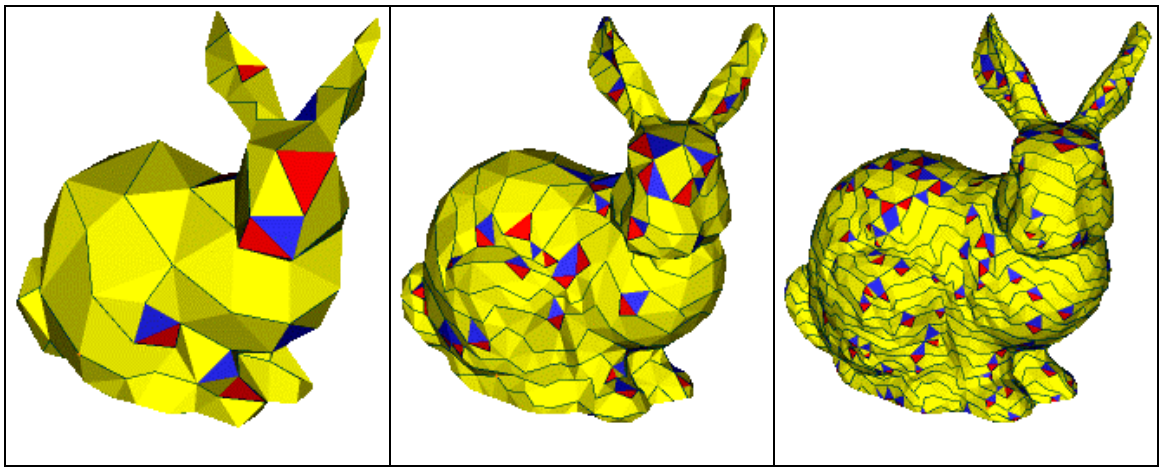


Figure 0-2 — A hierarchy of levels of detail

The hierarchical representation of a 3D model is based on the Progressive Forest Split scheme. In the Progressive Forest Split representation, a manifold 3D mesh is represented as a *base 3D mesh* followed by a sequence of *forest split* operations. The base 3D mesh is encoded as a single resolution 3D mesh using the Topological Surgery scheme. Each forest split operation is composed of a *forest* in the graph of the current 3D mesh, and a *sequence of simple polygons*. Figure 0-3 illustrates the method. To prevent visual artifacts which may occur while switching from a level of detail to the next one, the data structure supports smooth transition between consecutive levels of detail in the form of linear interpolation of vertex positions.

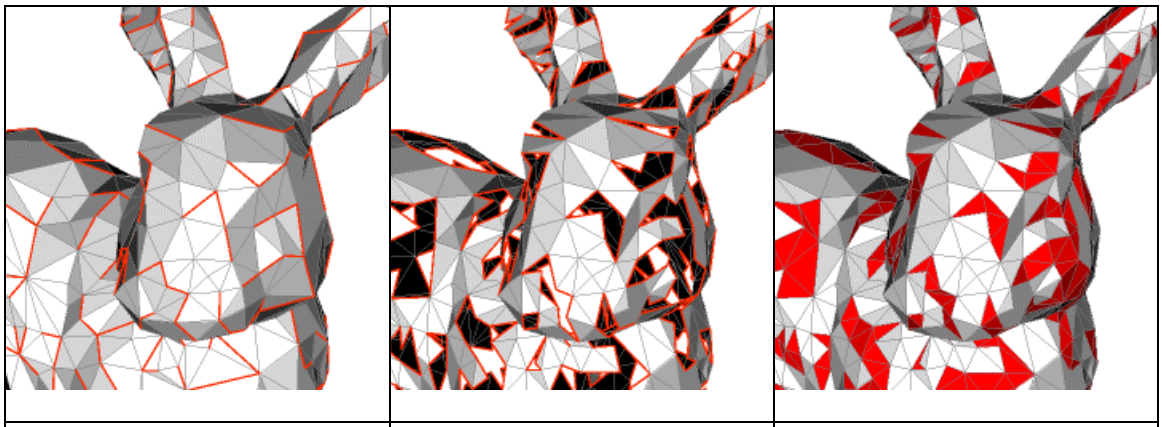


Figure 0-3 — The Forest Split operation. A 3D mesh with a forest of edges marked (A). The tree loops resulting from cutting the 3D mesh through the forest edges (B). Each tree loop is filled by a simple polygon composed of polygonal faces (C).

Error Resilience for 3D mesh object

If the 3D mesh is partitioned into independent parts, it may be possible to perform more efficient data transmission in an error-prone environment, e.g., an IP network or datacasting service in a broadcast TV network. It must be possible to resynchronize after a channel error, and continue data transmission and rendering from that point instead of starting over from scratch. Even with the presence of channel errors, the decoder can start decoding and rendering from the next partition that is received intact from the channel.

Flexible partitioning methods can be used to organize the data, such that it fits the underlying network packet structure more closely, and overhead is reduced to the minimum. To allow flexible partitioning, several connected components may be merged into one partition, where as a large connected component may be divided into several independent partitions. Merging and dividing of connected components using different partition types can be done at any point in the 3D mesh object.

© ISO/IEC 2004 – All rights reserved

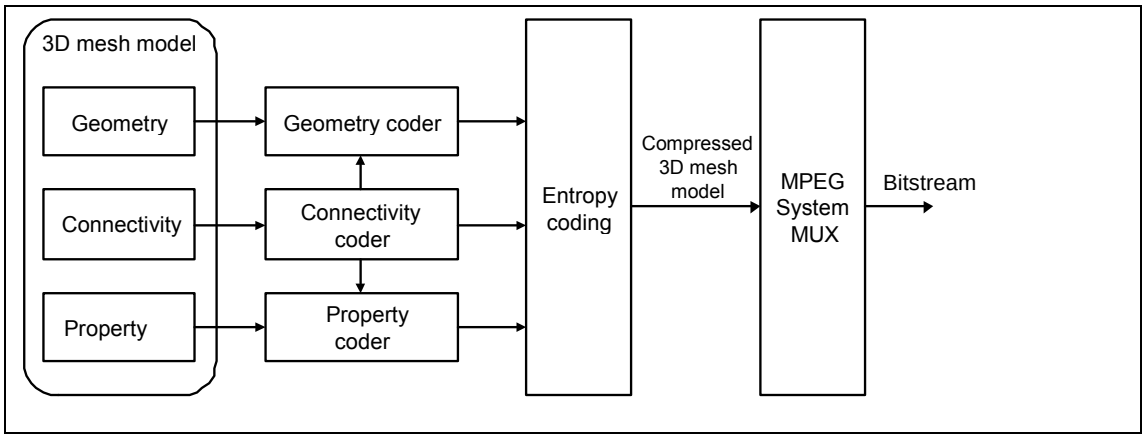
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Stitching for Non-Manifold and Non-Orientable Meshes

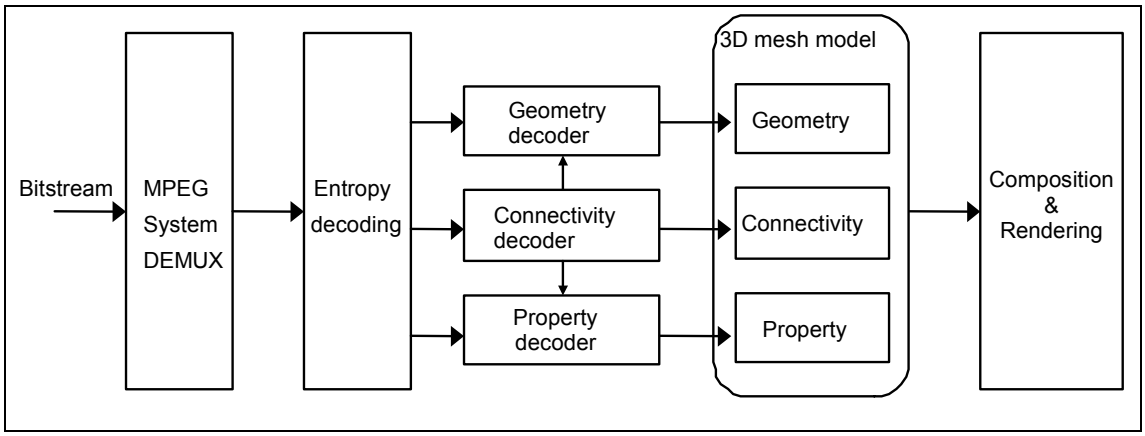
The connectivity of a non-manifold and non-orientable 3D mesh is represented as a manifold 3D mesh and a sequence of *stitches*. Each stitch describes how to identify one or more pairs of vertices along two linear paths of edges, and each one of these paths is contained in one of the vertex graphs that span the connected components of the manifold 3D mesh.

Encoder and Decoder Block Diagrams

High level block diagrams of a general 3D polygonal model encoder and decoder are shown in Figure 0-4. They consist of a 3D mesh connectivity (de)coder, geometry (de)coder, property (de)coder, and entropy (de)coding blocks. Connectivity, vertex position, and property information are extracted from 3D mesh model described in VRML or MPEG-4 BIFS format. The connectivity (de)coder is used for an efficient representation of the association between each face and its sustaining vertices. The geometry (de)coder is used for a lossy or lossless compression of vertex coordinates. The property (de)coder is used for a lossy or lossless compression of colour, normal, and texture coordinate data.



(a) 3D mesh encoder



(b) 3D mesh decoder

Figure 0-4 — General block diagram of the 3D mesh compres

Overview of the object based nonscalable syntax

The coded representation defined in the non-scalable syntax achieves a high compression ratio while preserving good image quality. Further, when access to individual objects is desired, the shape of objects also needs to be coded, and depending on the bandwidth available, the shape information can be coded in a lossy or lossless fashion.

The compression algorithm employed for texture data is not lossless as the exact sample values are not preserved during coding. Obtaining good image quality at the bitrates of interest demands very high compression, which is not

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

achievable with intra coding alone. The need for random access, however, is best satisfied with pure intra coding. The choice of the techniques is based on the need to balance a high image quality and compression ratio with the requirement to make random access to the coded bitstream.

A number of techniques are used to achieve high compression. The algorithm first uses block-based motion compensation to reduce the temporal redundancy. Motion compensation is used both for causal prediction of the current VOP from a previous VOP, and for non-causal, interpolative prediction from past and future VOPs. Motion vectors are defined for each 16-sample by 16-line region of a VOP or 8-sample by 8-line region of a VOP as required. The prediction error, is further compressed using the discrete cosine transform (DCT) to remove spatial correlation before it is quantised in an irreversible process that discards the less important information. Finally, the shape information, motion vectors and the quantised DCT information, are encoded using variable length codes.

In order to preserve the lossless quality, or to restrict the maximum bit count of block data, the block based DPCM coding can be used for Studio Profiles.

Temporal processing

Because of the conflicting requirements of random access to and highly efficient compression, three main VOP types are defined. Intra coded VOPs (I-VOPs) are coded without reference to other pictures. They provide access points to the coded sequence where decoding can begin, but are coded with only moderate compression. Predictive coded VOPs (P-VOPs) are coded more efficiently using motion compensated prediction from a past intra or predictive coded VOPs and are generally used as a reference for further prediction. Bidirectionally-predictive coded VOPs (B-VOPs) provide the highest degree of compression but require both past and future reference VOPs for motion compensation. Bidirectionally-predictive coded VOPs are never used as references for prediction (except in the case that the resulting VOP is used as a reference for scalable enhancement layer). The organisation of the three VOP types in a sequence is very flexible. The choice is left to the encoder and will depend on the requirements of the application.

Coding of Shapes

In natural video scenes, VOPs are generated by segmentation of the scene according to some semantic meaning. For such scenes, the shape information is thus binary (binary shape). Shape information is also referred to as alpha plane. The binary alpha plane is coded on a macroblock basis by a coder which uses the context information, motion compensation and arithmetic coding. For Studio Profiles particularly, HHC binary alpha block coding is used.

For coding of shape of a VOP, a bounding rectangle is first created and is extended to multiples of 16×16 blocks with extended alpha samples set to zero. Shape coding is then initiated on a 16×16 block basis; these blocks are also referred to as binary alpha blocks.

Coding interlaced video

Each frame of interlaced video consists of two fields which are separated by one field-period. Studio Profiles allow either the frame to be encoded as a VOP or the two fields to be encoded as two VOPs. Frame encoding or field encoding can be adaptively selected on a frame-by-frame basis. Frame encoding is typically preferred when the video scene contains significant detail with limited motion. Field encoding, in which the second field can be predicted from the first, works better when there is fast movement.

Motion representation - macroblocks

The choice of 16×16 blocks (referred to as macroblocks) for the motion-compensation unit is a result of the trade-off between the coding gain provided by using motion information and the overhead needed to represent it. Each macroblock can further be subdivided to 8×8 blocks for motion estimation and compensation depending on the overhead that can be afforded. In order to encode the highly active scene with higher VOP rate, a Reduced Resolution VOP tool is provided. When this tool is used , the size of the macroblock used for motion compensation decoding is 32x32 pixels and the size of block is 16x16 pixels.

For Studio Profiles particularly, in frame encoding, the prediction from the previous reference frame can itself be either frame-based or field-based.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Depending on the type of the macroblock, motion vector information and other side information is encoded with the compressed prediction error in each macroblock. The motion vectors are differenced with respect to a prediction value and coded using variable length codes. The maximum length of the motion vectors allowed is decided at the encoder. It is the responsibility of the encoder to calculate appropriate motion vectors. The specification does not specify how this should be done.

Spatial redundancy reduction

Both source VOPs and prediction errors VOPs have significant spatial redundancy. This part of ISO/IEC 14496 uses a block-based DCT method with optional visually weighted quantisation, and run-length coding. After motion compensated prediction or interpolation, the resulting prediction error is split into 8×8 blocks. These are transformed into the DCT domain where they can be weighted before being quantised. After quantisation many of the DCT coefficients are zero in value and so two-dimensional run-length and variable length coding is used to encode the remaining DCT coefficients efficiently.

Chrominance formats

This part of ISO/IEC 14496 currently supports the 4:2:0 chrominance format. Studio Profiles support the 4:2:2 and 4:4:4 chorominance formats in addition.

RGB color components

Studio Profiles support coding of RGB color components. The resolution of each component shall be identical when input data is treated as RGB color components.

Pixel depth

This part of ISO/IEC 14496 supports pixel depths between 4 and 12 bits in luminance and chrominance planes. IStudio Profiles support 8, 10 and 12 bits in luminance and chrominance or RGB planes.

Generalized scalability

The scalability tools in this part of ISO/IEC 14496 are designed to support applications beyond that supported by single layer video. The major applications of scalability include internet video, wireless video, multi-quality video services, video database browsing etc. In some of these applications, either normal scalabilities on picture basis such as those in ISO/IEC 13818-2:2000 may be employed or object based scalabilities may be necessary; both categories of scalability are enabled by this part of ISO/IEC 14496.

Although a simple solution to scalable video is the simulcast technique that is based on transmission/storage of multiple independently coded reproductions of video, a more efficient alternative is scalable video coding, in which the bandwidth allocated to a given reproduction of video can be partially re-utilised in coding of the next reproduction of video. In scalable video coding, it is assumed that given a coded bitstream, decoders of various complexities can decode and display appropriate reproductions of coded video. A scalable video encoder is likely to have increased complexity when compared to a single layer encoder. However, this part of ISO/IEC 14496 provides several different forms of scalabilities that address non-overlapping applications with corresponding complexities.

The basic scalability tools offered are temporal scalability and spatial scalability. Moreover, combinations of these basic scalability tools are also supported and are referred to as hybrid scalability. In the case of basic scalability, two layers of video referred to as the lower layer and the enhancement layer are allowed, whereas in hybrid scalability up to four layers are supported.

Object based Temporal scalability

Temporal scalability is a tool intended for use in a range of diverse video applications from video databases, internet video, wireless video and multiview/stereoscopic coding of video. Furthermore, it may also provide a migration path from current lower temporal resolution video systems to higher temporal resolution systems of the future.

Temporal scalability involves partitioning of VOPs into layers, where the lower layer is coded by itself to provide the basic temporal rate and the enhancement layer is coded with temporal prediction with respect to the lower layer.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

These layers when decoded and temporally multiplexed yield full temporal resolution. The lower temporal resolution systems may only decode the lower layer to provide basic temporal resolution whereas enhanced systems of the future may support both layers. Furthermore, temporal scalability has use in bandwidth constrained networked applications where adaptation to frequent changes in allowed throughput are necessary. An additional advantage of temporal scalability is its ability to provide resilience to transmission errors as the more important data of the lower layer can be sent over a channel with better error performance, whereas the less critical enhancement layer can be sent over a channel with poor error performance. Object based temporal scalability can also be employed to allow graceful control of picture quality by controlling the temporal rate of each video object under the constraint of a given bit-budget.

Object Spatial scalability

Spatial scalability is a tool intended for use in video applications involving multi quality video services, video database browsing, internet video and wireless video, i.e., video systems with the primary common feature that a minimum of two layers of spatial resolution are necessary. Spatial scalability involves generating two spatial resolution video layers from a single video source such that the lower layer is coded by itself to provide the basic spatial resolution and the enhancement layer employs the spatially interpolated lower layer and carries the full spatial resolution of the input video source.

Object based spatial scalability is composed of texture spatial scalability and shape spatial scalability. They can be used independently or together by its application. Binary shape spatial scalability is used for the applications that have 'binary shape only' mode as well as the applications of general object based spatial scalability.

An additional advantage of spatial scalability is its ability to provide resilience to transmission errors as the more important data of the lower layer can be sent over a channel with better error performance, whereas the less critical enhancement layer data can be sent over a channel with poor error performance. Further, it can also allow interoperability between various standards.

Hybrid scalability

There are a number of applications where neither the temporal scalability nor the spatial scalability may offer the necessary flexibility and control. This may necessitate use of temporal and spatial scalability simultaneously and is referred to as the hybrid scalability. Among the applications of hybrid scalability are wireless video, internet video, multiviewpoint/stereoscopic coding etc.

Error Resilience

This part of ISO/IEC 14496 provides error robustness and resilience to allow accessing of image or video information over a wide range of storage and transmission media. The error resilience tools developed for this part of ISO/IEC 14496 can be divided into three major categories. These categories include synchronization, data recovery, and error concealment. It should be noted that these categories are not unique to this part of ISO/IEC 14496, and have been used elsewhere in general research in this area. It is, however, the tools contained in these categories that are of interest, and where this part of ISO/IEC 14496 makes its contribution to the problem of error resilience.

Fine Granularity Scalability

Two profiles are developed in response to the growing need for a video coding method for Streaming Video on Internet applications. It provides the definition and description of Advanced Simple (AS) Profile and Fine Granularity Scalable (FGS) Profile. AS Profile provides the capability to distribute single-layer frame based video at a wide range of bit rates available for the distribution of video on Internet. FGS Profile uses AS Video Object in the base layer and provides the description of two enhancement layer types - Fine Granularity Scalability (FGS) and FGS Temporal Scalability (FGST). FGS Profile allows the coverage of a wide range of bit rates for the distribution of video on Internet with the flexibility of using multiple layers, where there is a wide range of bandwidth variation.

Fine Granularity Scalability (FGS) provides quality scalability for each VOP. Figure 0-5 shows a basic FGS decoder structure.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

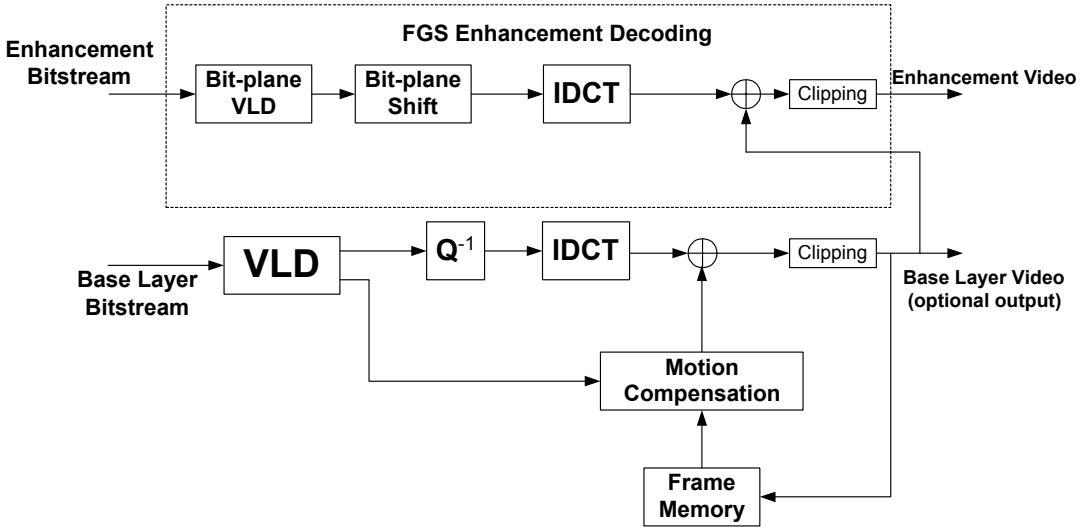


Figure 0-5 — A Basic FGS Decoder Structure

To reconstruct the enhanced VOP, the enhancement bitstream is first decoded using bit-plane VLD. The decoded block-bps are used to reconstruct the DCT coefficients in the DCT domain which are then right-shifted based on the frequency weighting and selective enhancement shifting factors. The output of bit-plane shift is the DCT coefficients of the image domain residues. After the IDCT, the image domain residues are reconstructed. They are added to the reconstructed clipped base-layer pixels to reconstruct the enhanced VOP. The reconstructed enhanced VOP pixels are limited into the value range between 0 and 255 by the clipping unit in the enhancement layer to generate the final enhanced video. The reconstructed base layer video is available as an optional output since each base layer reconstructed VOP needs to be stored in the frame buffer for motion compensation.

The basic FGS enhancement layer consists of FGS VOPs that enhance the quality of the base-layer VOPs as shown in Figure 0-6.

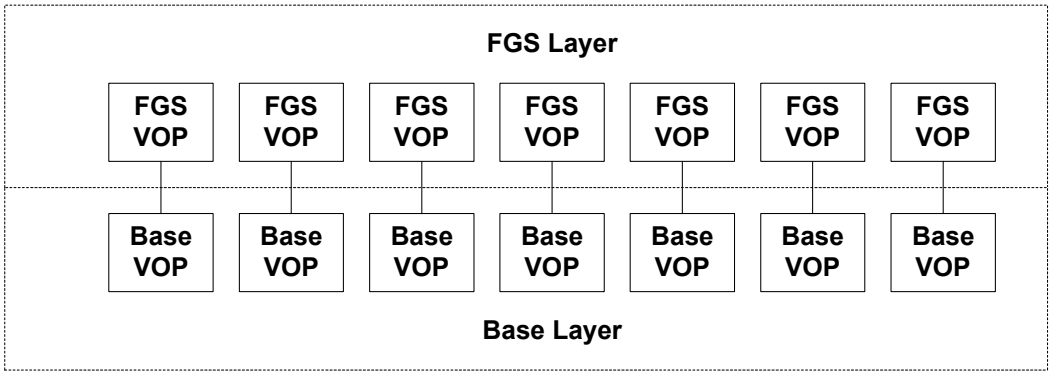


Figure 0-6 — Basic FGS Enhancement Structure

When FGS temporal scalability (FGST) is used, there are two possible enhancement structures. One structure is to have two separate enhancement layers for FGS and FGST as shown in Figure 0-7 and the other structure is to have one combined enhancement layer for FGS and FGST as shown in Figure 0-8.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

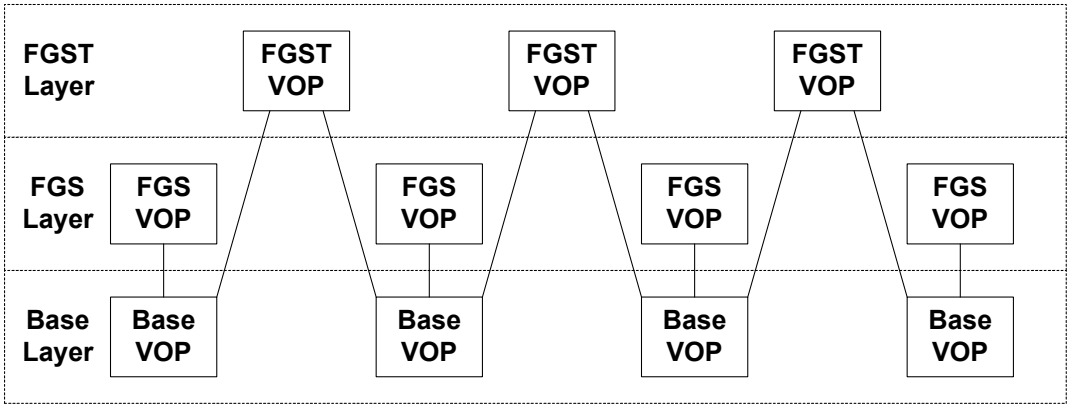


Figure 0-7 — Two Separate Enhancement Layers for FGS and FGST

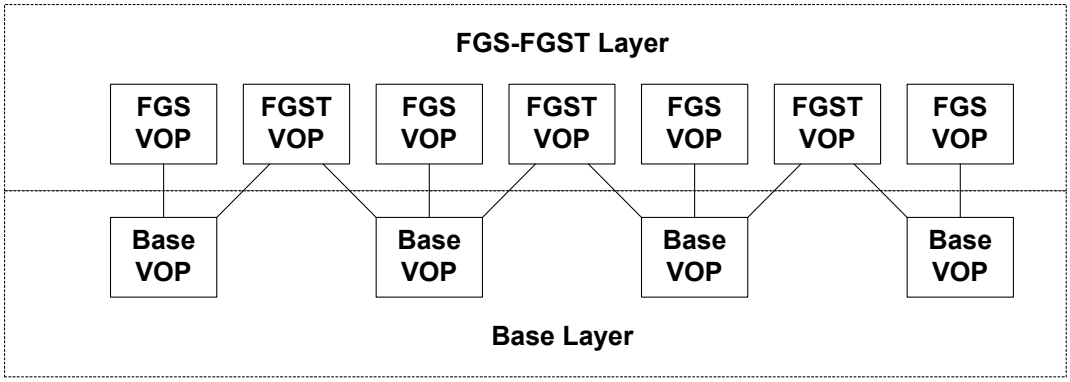


Figure 0-8 — One Combined Enhancement Layer for FGS and FGST

In either one of these two structures that include FGS temporal scalability, the prediction for the FGS temporal scalable VOPs can only be from the base layer. Each FGS temporal scalable VOP has two separate parts. The first part contains motion vector data and the second part contains the DCT texture data. The syntax for the first part is similar to that in the temporal scalability described in subclause 6.2. The DCT texture data in the second part are coded using bit-plane coding in the same way as that in FGS. To distinguish the temporal scalability in subclause 6.2 and FGS temporal scalability, the FGS temporal scalability layer in Figure 0-7 is called “FGST layer”. The combined FGS and FGST layer in Figure 0-8 is called “FGS-FGST layer”. The “FGS VOP” shown in Figure 0-7 and Figure 0-8 is an fgs vop with **fgs_vop_coding_type** being ‘I’. The “FGST VOP” shown in Figure 0-7 and Figure 0-8 is an fgs vop with **fgs_vop_coding_type** being ‘P’ or ‘B’.

The code value of **profile_and_level_indication** in VisualObjectSequence() has been extended to include the profile and level indications for AS Profile and FGS Profile. The identifier for an enhancement layer is the syntax **video_object_type_indication** in VideoObjectLayer(). A unique code is defined for FGS Object Type to indicate that this VOL contains fgs vops. Another unique code is defined for AS Object Type to indicate that this VOL is the base-layer. There is a syntax **fgs_layer_type** in VideoObjectLayer() to indicate whether this VOL is an FGS layer as shown in Figure 0-6 and Figure 0-7, or an FGST layer as shown in Figure 0-7, or an FGS-FGST layer as shown in Figure 0-8. Similar to the syntax structure in subclause 6.2, under each VOL for FGS, there is a hierarchy of fgs vop, fgs macroblock, and fgs block. An fgs vop starts with a unique **fgs_vop_start_code**. Within each fgs vop, there are multiple vop-bps. Each vop-bp in an fgs vop starts with an **fgs_bp_start_code** whose last 5 bits indicate the ID of the vop-bp. In each fgs macroblock, there are 4 block-bps for the luminance component (Y), 2 block-bps for the two chrominance components (U and V) for the 4:2:0 chrominance format. Each block-bp is coded by VLC.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Coding of audio-visual objects —

Part 2:
Visual

1 Scope

This part of ISO/IEC 14496 specifies the coded representation of picture information in the form of natural or synthetic visual objects like video sequences of rectangular or arbitrarily shaped pictures, moving 2D meshes, animated 3D face and body models and texture for synthetic objects. The coded representation allows for content based access for digital storage media, digital video communication and other applications. ISO/IEC 14496 specifies also the decoding process of the aforementioned coded representation. The representation supports constant bitrate transmission, variable bitrate transmission, robust transmission, content based random access (including normal random access), object based scalable decoding (including normal scalable decoding), object based bitstream editing, as well as special functions such as fast forward playback, fast reverse playback, slow motion, pause and still pictures. Synthetic objects and coding of special 2D/3D meshes, texture, and animation parameters are provided for use with downloadable models to exploit mixed media and the bandwidth improvement associated with remote manipulation of such models. ISO/IEC 14496 is intended to allow some level of interoperability with ISO/IEC 11172-2:1993, ISO/IEC 13818-2:2000 and ITU-T Recommendation H.263.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 11172-2:1993, *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 2: Video*

ISO/IEC 13818-1:2000, *Information technology — Generic coding of moving pictures and associated audio information: Systems*

ISO/IEC 13818-2:2000, *Information technology — Generic coding of moving pictures and associated audio information: Video*

IEEE Standard Specifications for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, IEEE Std 1180-1990, December 6, 1990

ITU-R Recommendation BT.601-5, *Studio encoding parameters of digital television for studios*

ITU-T Recommendation H.261(1993*), Codec for audiovisual services at px64 kbit/s*

ITU-T Recommendation H.263(2000), *Video Coding for Low Bitrate Communication*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1 **3D mesh**: A polygonal mesh. It is defined by the position of its vertices (geometry), by the association between each face and its sustaining vertices (connectivity), and optionally by colors, normals, and texture coordinates (properties). Properties do not affect the 3D geometry, but influence the way the model is shaded.

3.2 **3D mesh connectivity**: The association between each face of a 3D mesh and its sustaining vertices.

3.3 **3D mesh geometry**: The three dimensional coordinates of the vertices of a 3D mesh.

3.4 **3D mesh properties**: The normals, colors, and texture coordinates of a 3D mesh.

3.5 **AC coefficient**: Any DCT coefficient for which the frequency in one or both dimensions is non-zero.

3.6 **B-field VOP**: A field structure B-VOP.

3.7 **B-frame VOP**: A frame structure B-VOP.

3.8 **B-VOP**; **bidirectionally predictive-coded video object plane (VOP)**: A VOP that is coded using motion compensated prediction from past and/or future reference VOPs.

3.9 **backward motion vector**: A motion vector that is used for motion compensation from a reference VOP at a later time in display order.

3.10 **backward prediction**: Prediction from the future reference VOP.

3.11 **base layer**: An independently decodable layer of a scalable hierarchy.

3.12 **binary alpha block**: A block of size 16x16 pels, colocated with macroblock, representing shape information of the binary alpha map; it is also referred to as a bab.

3.13 **binary alpha map**: A 2D binary mask used to represent the shape of a video object such that the pixels that are opaque are considered as part of the object where as pixels that are transparent are not considered to be part of the object.

3.14 **bitstream; stream**: An ordered series of bits that forms the coded representation of the data.

3.15 **bitrate**: The rate at which the coded bitstream is delivered from the storage medium or network to the input of a decoder.

3.16 **block**: An 8-row by 8-column (or 16-row by 16-column in motion compensation decoding of Reduced Resolution VOP) matrix of samples, or 64 DCT coefficients (source, quantised or inverse quantised).

3.17 **block-bp**: An array of 64 bits, one from each DCT coefficient at the same significant position of accuracy in a zigzag scan order. When frequency weighting is used, block-bps are formed after the weighting is applied to the DCT coefficients in an 8x8 block.

3.18 **body animation parameters, BAP**: Coded streaming animation parameters that manipulate the body joints, and that govern deformations of the body surface with the use of BAT.

3.19 **body definition parameters, BDP**: Downloadable data to customize a baseline body model in the decoder to a particular body, or to download a body model along with the information about how to animate it. The BDPs are normally transmitted once per session, followed by a stream of compressed BAPs. BDPs may include body geometry, body skeleton, and body deformation tables.

3.20 **body deformation table, BodyDefTable**: A downloadable function mapping from incoming BAPs to body surface geometry that provides a combination of BAPs for controlling body surface geometry deformation.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.21 **3.20 bottom field**: One of two fields that comprise a frame. Each line of a bottom field is spatially located immediately below the corresponding line of the top field.

3.22 **boundary (of a 3D mesh)**: The set of boundary edges of a 3D mesh.

3.23 **boundary edge (of a 3D mesh)**: An edge of a 3D mesh which has exactly one incident face.

3.24 **bounding loop:** The boundary of a simple polygon.

3.25 **bounding loop index**: An index of the bounding loop table which contains the list of vertices lying on the bounding loop

3.26 **branching triangle (of a triangle tree)**: A triangle of a triangle tree with three incident triangle tree edges.

3.27 **branching vertex (of a vertex tree or of a vertex graph)**: A vertex of a vertex tree with three or more incident vertex tree edges.

3.28 **byte aligned**: A bit in a coded bitstream is byte-aligned if its position is a multiple of 8-bits from the first bit in the stream.

3.29 **byte**: Sequence of 8-bits.

3.30 **context based arithmetic encoding**: The method used for coding of binary shape; it is also referred to as cae.

3.31 **channel**: A digital medium or a network that stores or transports a bitstream constructed according to ISO/IEC 14496.

3.32 **chrominance format**: Defines the number of chrominance blocks in a macroblock.

3.33 **chrominance component**: A matrix, block or single sample representing one of the two colour difference signals related to the primary colours in the manner defined in the bitstream. The symbols used for the chrominance signals are Cr and Cb.

3.34 **coded B-frame**: A B-frame VOP or a pair of B-field VOPs that is coded.

3.35 **coded B-VOP**: A B-VOP that is coded.

3.36 **coded frame**: A coded frame is a coded I-frame, a coded P-frame or a coded B-frame.

3.37 **coded VOP**: A coded VOP is a coded I-VOP, a coded P-VOP, a coded B-VOP, or a coded S(GMC)-VOP.

3.38 **coded I-frame**: An I-frame VOP or a pair of field VOPs that is coded where the first field VOP is an I-VOP and the second field VOP is an I-VOP or a P-VOP.

3.39 **coded I-VOP**: An I-VOP that is coded.

3.40 **coded P-frame**: A P-frame VOP or a pair of field VOPs that is coded.

3.41 **coded P-VOP**: A P-VOP that is coded.

3.42 **coded S(GMC)-VOP**: A S(GMC)-VOP that is coded.

3.43 **coded order**: The order in which the VOPs are transmitted and decoded. This order is not necessarily the same as the display order.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.44 **coded video bitstream**: A coded representation of a series of one or more VOPs as defined in this part of ISO/IEC 14496.

3.45 **coded representation**: A data element as represented in its encoded form.

3.46 **coding parameters**: The set of user-definable parameters that characterise a coded video bitstream. Bitstreams are characterised by coding parameters. Decoders are characterised by the bitstreams that they are capable of decoding.

3.47 **component**: A matrix, block or single sample from one of the three matrices (luminance and two chrominance or green, blue and red color primaries) that make up a picture.

3.48 **composition process**: The (non-normative) process by which reconstructed VOPs are composed into a scene and displayed.

3.49 **compression**: Reduction in the number of bits used to represent an item of data.

3.50 **connected component (of a 3D mesh)**: A 3D mesh can be decomposed into one or more disjoint sets called connected components. Every face of the 3D mesh is assigned to an unique connected component, such that any pair of two faces that share an edge belongs to the same connected component.

3.51 **constant bitrate coded video**: A coded video bitstream with a constant bitrate.

3.52 **constant bitrate**: Operation where the bitrate is constant from start to finish of the coded bitstream.

3.53 **conversion ratio**: The size conversion ratio for the purpose of rate control of shape.

3.54 **corner (of a 3D mesh)**: A (face,vertex) pair, where the vertex is one of the sustaining vertices of the face.

3.55 **data element**: An item of data as represented before encoding and after decoding.

3.56 **DC coefficient**: The DCT coefficient for which the frequency is zero in both dimensions.

3.57 **DCT coefficient**: The amplitude of a specific cosine basis function.

3.58 **decoder input buffer**: The first-in first-out (FIFO) buffer specified in the video buffering verifier.

3.59 **decoder**: An embodiment of a decoding process.

3.60 **decoding order**: The order in which the VOPs are transmitted and decoded. This order is not necessarily the same as the display order.

3.61 **decoding (process)**: The process defined in this part of ISO/IEC 14496 that reads an input coded bitstream and produces decoded VOPs or audio samples, or 3D meshes.

3.62 **digital storage media; DSM**: A digital storage or transmission device or system.

3.63 **discrete cosine transform; DCT**: Either the forward discrete cosine transform or the inverse discrete cosine transform. The DCT is an invertible, discrete orthogonal transformation. The inverse DCT is defined in Annex A.

3.64 **3.64 display aspect ratio**: The ratio height/width (in spatial measurement units such as centimeters) of the intended display.

3.65 **display order**: The order in which the decoded pictures are displayed. Normally this is the same order in which they were presented at the input of the encoder.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.66 **display process**: The (non-normative) process by which reconstructed frames are displayed.

3.67 **dual graph (of a 3D mesh)**: A graph composed of a set of *dual graph nodes* and a set of *dual graph edges*. Each dual graph node corresponds to a face of the 3D mesh. Each dual graph edge corresponds to an internal edge of the 3D mesh and links two dual graph nodes.

3.68 **edge (of a 3D mesh)**: An unordered pair of vertex indices of a 3D mesh, consecutive in one or more faces of the 3D mesh (modulo cyclical permutations of the sequences).

3.69 **editing**: The process by which one or more coded bitstreams are manipulated to produce a new coded bitstream. Conforming edited bitstreams must meet the requirements defined in this part of ISO/IEC 14496.

3.70 **encoder**: An embodiment of an encoding process.

3.71 **encoding (process)**: A process, not specified in this part of ISO/IEC 14496, that reads a stream of input pictures or audio samples and produces a valid coded bitstream as defined in this part of ISO/IEC 14496.

3.72 **end of plane; eop**: A symbol to indicate whether a '1' bit is the last 1' bit of a block-bp.

3.73 **enhancement layer**: A relative reference to a layer (above the base layer) in a scalable hierarchy. For all forms of scalability, its decoding process can be described by reference to the lower layer decoding process and the appropriate additional decoding process for the enhancement layer itself.

3.74 **face (of a 3D mesh)**: An alternating sequence of vertices and edges, such that each edge in the sequence connects the vertices immediately preceding and following it in the sequence (with the exception of the last edge that connects the vertex preceding it with the first vertex of the sequence). No vertex may appear more than once in the sequence. In practice, a face is often represented by the list of vertices in the sequence.

3.75 **global motion compensation, GMC**: The use of global spatial transformation to improve the efficiency of the prediction of sample values. The prediction uses global spatial transformation to provide offsets into the past reference VOPs containing previously decoded sample values that are used to form the prediction error.

3.76 **face animation parameter units, FAPU**: Special normalized units (e.g. translational, angular, logical) defined to allow interpretation of FAPs with any facial model in a consistent way to produce reasonable results in expressions and speech pronunciation.

3.77 **face animation parameters, FAP**: Coded streaming animation parameters that manipulate the displacements and angles of face features, and that govern the blending of visemes and face expressions during speech.

3.78 **face animation table, FAT**: A downloadable function mapping from incoming FAPs to feature control points in the face mesh that provides piecewise linear weightings of the FAPs for controlling face movements.

3.79 **face calibration mesh**: Definition of a 3D mesh for calibration of the shape and structure of a baseline face model.

3.80 **face definition parameters, FDP**: Downloadable data to customize a baseline face model in the decoder to a particular face, or to download a face model along with the information about how to animate it. The FDPs are normally transmitted once per session, followed by a stream of compressed FAPs. FDPs may include feature points for calibrating a baseline face, face texture and coordinates to map it onto the face, animation tables, etc.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.81 **face feature control point**: A normative vertex point in a set of such points that define the critical locations within face features for control by FAPs and that allow for calibration of the shape of the baseline face.

3.82 **face forest (of a 3D mesh)**: The set of all the face trees corresponding to the connected components of a 3D mesh.

3.83 **face interpolation transform, FIT**: A downloadable node type defined in ISO/IEC 14496-1:2001 for optional mapping of incoming FAPs to FAPs before their application to feature points, through weighted rational polynomial functions, for complex cross-coupling of standard FAPs to link their effects into custom or proprietary face models.

3.84 **face model mesh**: A 2D or 3D contiguous geometric mesh defined by vertices and planar polygons utilizing the vertex coordinates, suitable for rendering with photometric attributes (e.g. texture, color, normals).

3.85 **face tree (of a 3D mesh)**: A spanning tree constructed in the dual graph of a connected component.

3.86 **fast forward playback**: The process of displaying a sequence, or parts of a sequence, of VOPs in display order, faster than real-time.

3.87 **fast reverse playback**: The process of displaying a sequence, or parts of a sequence, of VOPs in the reverse of display order, faster than real-time.

3.88 **feathering**: A tool that tapers the values around edges of binary alpha mask for composition with the background.

3.89 **fgs block**: An 8-row by 8-column matrix of bits, each from one DCT coefficient at the same significant position of accuracy, or its coded representation. The usage is clear from the context.

3.90 **fgs macroblock**: The four block-bps of luminance component (Y) and the two (for 4:2:0 chrominance format) corresponding block-bps of chrominance components (U and V) with the same accuracy significance coming from the DCT coefficients of a macroblock. It may also be used to refer to the coded representation of the six block-bps. The usage is clear from the context.

3.91 **fgs macroblock number**: A number for an fgs macroblock within a vop-bp. The fgs macroblock number of the top-left fgs macroblock in each vop-bp shall be zero. The fgs macroblock number increments from left to right and from top to bottom.

3.92 **fgs run**: The number of '0' bits preceding a '1' bit within a block-bp.

3.93 **fgs temporal scalability; FGST**: A type of scalability where an enhancement layer uses predictions from sample data derived from the base layer using motion vectors. The VOP size in the enhancement layer is the same as that in the base layer. FGST is a specific type of temporal scalability where all DCT coefficients are coded using bit-plane coding as in FGS.

3.94 **fgs vop**: The pixel differences between the original VOP and the reconstructed VOP in the base layer. It may be used to refer to the DCT coefficients of the pixel differences or the original VOP. It may also be used to refer to the coded representation of the DCT coefficients. In the context of FGST, fgs vop refers to the original temporal scalable VOP. The usage is clear from the context.

3.95 **fine granularity scalability; FGS**: A type of scalability where an enhancement layer uses prediction from sample data of reconstructed VOP in the base layer. The encoded bitstream for each fgs vop can be truncated into any number of bits. The truncated bitstream for each fgs vop can be decoded to provide quality enhancement proportional to the amount of bits in the truncated bitstream of the fgs vop. The fgs vop has the same size and VOP rate as those of the base layer.

3.96 **field**: For an interlaced video signal, a "field" is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.97 **field-based prediction**: A prediction mode using only one field of the reference frame. The predicted block size is 16x16 luminance samples. Field-based prediction is not used in progressive frames.

3.98 **field period**: The reciprocal of twice the frame rate.

3.99 **field VOP; field structure VOP**: A field structure VOP is a coded VOP with vop_structure is equal to "Top field" or "Bottom field".

3.100 **flag**: A one bit integer variable which may take one of only two values (zero and one).

3.101 **forbidden**: The term "forbidden" when used in the clauses defining the coded bitstream indicates that the value shall never be used. This is usually to avoid emulation of start codes.

3.102 **forced updating**: The process by which macroblocks are intra-coded from time-to-time to ensure that mismatch errors between the inverse DCT processes in encoders and decoders cannot build up excessively.

3.103 **forward compatibility**: A newer coding standard is forward compatible with an older coding standard if decoders designed to operate with the newer coding standard are able to decode bitstreams of the older coding standard.

3.104 **forward motion vector**: A motion vector that is used for motion compensation from a reference frame VOP at an earlier time in display order.

3.105 **forward prediction**: Prediction from the past reference VOP.

3.106 **frame**: A frame contains lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the frame. For interlaced video a frame consists of two fields, a top field and a bottom field. One of these fields will commence one field period later than the other.

3.107 **frame-based prediction**: A prediction mode using both fields of the reference frame.

3.108 **frame period**: The reciprocal of the frame rate.

3.109 **frame rate**: The rate at which frames are be output from the composition process.

3.110 **frame VOP; frame structure VOP**: A frame structure VOP is a coded VOP with vop_structure is equal to "Frame".

3.111 **future reference frame (field)**: A future reference frame (field) is a reference frame (field) that occurs at a later time than the current VOP in display order.

3.112 **future reference VOP**: A future reference VOP is a reference VOP that occurs at a later time than the current VOP in display order.

3.113 **hierarchical 3D mesh**: A BIFS node representing a 3D mesh.

3.114 **hierarchical representation (of a 3D mesh)**: A 3D mesh representation with two or more levels of detail in increasing order of resolution, composed of a first coarse level of detail (base 3D mesh) followed by one or more refinement operations. Each refinement operation determines how to construct a new level of detail as a refinement of the current level of detail.

3.115 **hybrid scalability**: Hybrid scalability is the combination of two (or more) types of scalability.

3.116 **I-field VOP**: A field structure I-VOP.

3.117 **I-frame VOP**: A frame structure I-VOP.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.118 **imaginary edge**: An imaginary edge is inserted to triangulate a polygon.

3.119 **incremental representation (of a 3D mesh)**: A representation that allows the terminal to start rendering a subset of the 3D mesh before the whole bitstream has been completely received and/or decoded.

3.120 **IndexedFaceSet**: A BIFS node representing a 3D mesh.

3.121 **interlace**: The property of conventional television frames where alternating lines of the frame represent different instances in time. In an interlaced frame, one of the field is meant to be displayed first. This field is called the first field. The first field can be the top field or the bottom field of the frame.

3.122 **internal edge (of a 3D mesh)**: An edge of a 3D mesh which has exactly two incident faces.

3.123 **I-VOP; intra-coded VOP**: A VOP coded using information only from itself.

3.124 **intra coding**: Coding of a macroblock or VOP that uses information only from that macroblock or VOP.

3.125 **intra shape coding**: Shape coding that does not use any temporal prediction.

3.126 **inter shape coding**: Shape coding that uses temporal prediction.

3.127 **inverse quantisation**: The process of rescaling the quantised DCT coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse DCT.

3.128 **jump edge (of a 3D mesh)**: An edge of the vertex graph which does not belong to any vertex tree.

3.129 **level**: A defined set of constraints on the values which may be taken by the parameters of this part of ISO/IEC 14496 within a particular profile. A profile may contain one or more levels. In a different context, level is the absolute value of a non-zero coefficient (see “run”).

3.130 **layer**: In a scalable hierarchy denotes one out of the ordered set of bitstreams and (the result of) its associated decoding process.

3.131 **layered bitstream**: A single bitstream associated to a specific layer (always used in conjunction with layer qualifiers, e. g. “enhancement layer bitstream”).

3.132 **leaf triangle (of a 3D mesh)**: A triangle of a triangle tree with exactly one incident triangle tree edge.

3.133 **leaf vertex (of a 3D mesh)**: A vertex of a vertex tree with exactly one incident vertex tree edge.

3.134 **lower layer**: A relative reference to the layer immediately below a given enhancement layer (implicitly including decoding of *all* layers below this enhancement layer).

3.135 **luminance component**: A matrix, block or single sample representing a monochrome representation of the signal and related to the primary colours in the manner defined in the bitstream. The symbol used for luminance is Y.

3.136 **Mbit**: 1 000 000 bits.

3.137 **macroblock**: The four 8×8 (or 16x16 in motion compensation decoding of Reduced Resolution VOP) blocks of luminance data and the two (for 4:2:0 chrominance format) or four (for 4:2:2 chrominance format), eight (for 4:4:4 chrominance format) corresponding 8×8 (or 16x16 in motion compensation decoding of Reduced Resolution VOP) blocks of chrominance data coming from a 16×16 (or 32x32 in motion compensation decoding of Reduced Resolution VOP) section of the luminance component of the picture. Macroblock is sometimes used to refer to the sample data and sometimes to the coded representation of the sample values and other data elements defined in the macroblock header of the syntax defined in this part of ISO/IEC 14496. The usage is clear from the context.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.138 **manifold (3D mesh)**: A 3D mesh which has neither singular vertices nor singular edges.

3.139 **marching edge (of a 3D mesh)**: An edge shared by two triangles along a triangle run.

3.140 **mesh (2D)**: A 2D triangular mesh refers to a planar graph which tessellates a video object plane into triangular patches. The vertices of the triangular mesh elements are referred to as node points. The straight-line segments between node points are referred to as edges. Two triangles are adjacent if they share a common edge.

3.141 **mesh geometry**: The spatial locations of the node points and the triangular structure of a mesh.

3.142 **mesh motion**: The temporal displacements of the node points of a mesh from one time instance to the next.

3.143 **mesh object plane**, **MOP**: The instance of a mesh object at a given time.

3.144 **motion compensation**: The use of motion vectors to improve the efficiency of the prediction of sample values. The prediction uses motion vectors to provide offsets into the past and/or future reference VOPs containing previously decoded sample values that are used to form the prediction error.

3.145 **motion estimation**: The process of estimating motion vectors during the encoding process.

3.146 **motion vector**: A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture or field to the coordinates in a reference VOP.

3.147 **motion vector for shape**: A motion vector used for motion compensation of shape.

3.148 **multiple auxiliary components**: Auxiliary components are defined for the VOP on a pixel-by-pixel basis, and contain data related to the video object, such as disparity, depth, and additional texture. Up to three auxiliary components (including the grayscale shape) are possible.

3.149 **non-intra coding**: Coding of a macroblock or a VOP that uses information both from itself and from macroblocks and VOPs occurring at other times.

3.150 **non-manifold (3D mesh)**: A 3D mesh that contains singular vertices or singular edges.

3.151 **non-orientable (3D mesh)**: A manifold 3D mesh which is not orientable.

3.152 **opaque macroblock**: A macroblock with shape mask of all 255's.

3.153 **orientable (3D mesh)**: A manifold 3D mesh for which an orientation can be chosen for each of its faces such that for each internal edge, the two faces incident to the edge induce opposite orientations on the common edge. All non-manifold 3D meshes are non-orientable.

3.154 **orientation (of a face of a 3D mesh)**: One of the two possible cyclical orderings of its sustaining vertices.

3.155 **orientation (of an edge of a 3D mesh)**: One of the two possible orderings of the two vertices of the edge.

3.156 **oriented (manifold 3D mesh)**: A manifold 3D mesh for which an orientation has been chosen for each of its faces such that for each internal edge, the two faces incident to the edge induce opposite orientations on the common edge. Note that a manifold 3D mesh may be orientable but not oriented. Such a mesh can be oriented by choosing a consistent orientation for its faces (which may require inverting the orientation of one or more faces).

3.157 **P-VOP; predictive-coded VOP**: A picture that is coded using motion compensated prediction from the past VOP.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.158 **parameter**: A variable within the syntax of this part of ISO/IEC 14496 which may take one of a range of values. A variable which can take one of only two values is called a flag.

3.159 **partition**: a bitstream segment of a 3D mesh that can be decoded independently.

3.160 **past reference picture**: A past reference VOP is a reference VOP that occurs at an earlier time than the current VOP in display order.

3.161 **P-field VOP**: A field structure P-VOP.

3.162 **P-frame VOP**: A frame structure P-VOP.

3.163 **picture**: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals or rgb colour signals. A “coded VOP” was defined earlier. For progressive video, a picture is identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.

3.164 **prediction**: The use of a predictor to provide an estimate of the sample value or data element currently being decoded.

3.165 **prediction error**: The difference between the actual value of a sample or data element and its predictor.

3.166 **predictor**: A linear combination of previously decoded sample values or data elements.

3.167 **profile**: A subset of the syntax of this part of ISO/IEC 14496, defined in terms of Visual Object Types.

3.168 **progressive**: The property of film frames where all the samples of the frame represent the same instances in time.

3.169 **quantisation matrix**: A set of sixty-four 8-bit values used by the inverse quantisation process.

3.170 **quantised DCT coefficients**: DCT coefficients before inverse quantisation. A variable length coded representation of quantised DCT coefficients is transmitted as part of the coded video bitstream.

3.171 **quantiser scale**: A scale factor coded in the bitstream and used by the decoding process to scale the inverse quantisation.

3.172 **random access**: The process of beginning to read and decode the coded bitstream at an arbitrary point.

3.173 **reconstructed VOP**: A reconstructed VOP consists of three matrices of 8-bit numbers representing the luminance and two chrominance signals. It is obtained by decoding a coded VOP.

3.174 **reference layer**: A layer to be referenced for prediction in a scalable hierarchy. The video_object_id of the reference layer should be the same value as the video_object_id of the enhancement layer. The ref_layer_id of the enhancement layer is set to the same value as the video_object_layer_id of the reference layer.

3.175 **reference VOP**: A reference VOP is a reconstructed VOP that was coded in the form of a coded I-VOP, a coded P-VOP, or a coded S(GMC)-VOP. Reference VOPs are used for forward and backward prediction when P-VOPs, B-VOPs, and S(GMC)-VOPs are decoded.

3.176 **regular triangle (of a triangle tree)**: A triangle of a triangle tree with exactly two incident triangle tree edges.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2004 – All rights reserved
Not for Resale

3.177 **regular vertex (of a 3D mesh)**: A vertex of a 3D mesh is *regular* if the set of incident edges and incident faces can be ordered in an alternating sequence of edges and faces, such that each face in the sequence is bounded by the edges immediately preceeding and following it in the sequence.

3.178 **regular vertex (of a vertex tree)**: A vertex of a vertex tree with exactly two incident vertex tree edges.

3.179 **reordering delay**: A delay in the decoding process that is caused by VOP reordering.

3.180 **reserved**: The term “reserved” when used in the clauses defining the coded bitstream indicates that the value may be used in the future for ISO/IEC defined extensions.

3.181 **RGB component**: A matrix, block or single sample representing one of the three primary colours. The symbols used for the rgb signals are Green, Blue and Red.

3.182 **sample aspect ratio**: (abbreviated to **SAR**). This specifies the relative distance between samples. It is defined (for the purposes of this specification) as the vertical displacement of the lines of luminance samples in a frame divided by the horizontal displacement of the luminance samples. Thus its units are (metres per line) ÷ (metres per sample)

3.183 **scalable hierarchy**: coded video data consisting of an ordered set of more than one video bitstream.

3.184 **scalability**: Scalability is the ability of a decoder to decode an ordered set of bitstreams to produce a reconstructed sequence. Moreover, useful video is output when subsets are decoded. The minimum subset that can thus be decoded is the first bitstream in the set which is called the base layer. Each of the other bitstreams in the set is called an enhancement layer. When addressing a specific enhancement layer, “lower layer” refers to the bitstream that precedes the enhancement layer.

3.185 **side information**: Information in the bitstream necessary for controlling the decoder.

3.186 **skipped macroblock**: A macroblock for which no data is encoded.

3.187 **run**: The number of zero coefficients preceding a non-zero coefficient, in the scan order. The absolute value of the non-zero coefficient is called “level”.

3.188 **S-VOP**: A picture that is coded using information obtained by warping whole or part of a static sprite.

3.189 **S(GMC)-VOP**: A picture that is coded using prediction based on global motion compensation from the past VOP.

3.190 **saturation**: Limiting a value that exceeds a defined range by setting its value to the maximum or minimum of the range as appropriate.

3.191 **simple polygon (of a 3D mesh)**: A connected component of which the dual graph is a tree. A simple polygon has exactly one boundary, and all its vertices lie on the boundary.

3.192 **singular edge (of a 3D mesh)**: An edge of a 3D mesh which has three or more incident faces.

3.193 **singular vertex (of a 3D mesh)**: A vertex of a 3D mesh which is not a regular vertex.

3.194 **source**; **input**: Term used to describe the video material or some of its attributes before encoding.

3.195 **spanning forest (of a graph)**: A subgraph of a graph composed of one or more trees, each tree spanning a connected component of the graph.

3.196 **spatial prediction**: prediction derived from a decoded frame of the reference layer decoder used in spatial scalability.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.197 **spatial scalability**: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer without using motion vectors. The layers can have different VOP sizes or VOP rates.

3.198 **static sprite**: The luminance, chrominance and binary alpha plane for an object which does not vary in time.

3.199 **start codes**: 32-bit codes embedded in that coded bitstream that are unique. They are used for several purposes including identifying some of the structures in the coding syntax.

3.200 **stuffing (bits)**; **stuffing (bytes)**: Code-words that may be inserted into the coded bitstream that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream which would otherwise be lower than the desired bitrate.

3.201 **temporal prediction**: prediction derived from reference VOPs other than those defined as spatial prediction.

3.202 **temporal scalability**: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer using motion vectors. The layers have identical frame size, and but can have different VOP rates.

3.203 **top field**: One of two fields that comprise a frame. Each line of a top field is spatially located immediately above the corresponding line of the bottom field.

3.204 **top layer**: the topmost layer (with the highest layer_id) of a scalable hierarchy.

3.205 **transparent macroblock**: A macroblock with shape mask of all zeros.

3.206 **traversal order**: The traversal order determines which branch to be traversed first at each branching triangle.

3.207 **tree loop (of a 3D mesh)**: An alternating sequence of vertices and edges bounding the hole created by cutting a 3D mesh through a tree of edges within a forest split operation.

3.208 **tree loop face**: a face incident to a tree loop.

3.209 **tree loop corner**: a corner of a tree loop face incident to a tree loop edge.

3.210 **triangle tree (of a 3D mesh)**: A spanning tree in the dual graph of a connected component of a triangular 3D mesh.

3.211 **triangle run**: A path in a triangle tree connecting a first leaf or branching triangle to a last leaf or branching triangle through zero or more intermediate triangles.

3.212 **triangular 3D mesh**: A 3D mesh with triangular faces. Also, a 3D mesh after all the polygonal faces have been triangulated.

3.213 **variable bitrate**: Operation where the bitrate varies with time during the decoding of a coded bitstream.

3.214 **variable length coding**; **VLC**: A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.

3.215 **vertex graph (of a 3D mesh)**: The graph formed by the vertices and the edges of the 3D mesh which do not correspond to dual graph edges included in the face forest.

3.216 **vertex run**: A path in a vertex tree connecting a first leaf or branching vertex to a last leaf or branching vertex through zero or more intermediate regular vertices.

--`,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.217 **vertex tree**: A rooted tree spanning one connected component of the vertex graph.

3.218 **video buffering verifier**; **VBV**: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

3.219 **video complexity verifier**; **VCV**: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the maximum processing requirements of the bitstream that an encoder or editing process may produce.

3.220 **video memory verifier**; **VMV**: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the maximum reference memory requirements of the bitstream that an encoder or editing process may produce.

3.221 **video object plane**, **VOP**: The instance of a video object at a given time.

3.222 **video presentation verifier**; **VPV**: Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the maximum presentation memory requirements of the bitstream that an encoder or editing process may produce.

3.223 **video session**: The highest syntactic structure of coded video bitstreams. It contains a series of one or more coded video objects.

3.224 **viseme**: the physical (visual) configuration of the mouth, tongue and jaw that is visually correlated with the speech sound corresponding to a phoneme.

3.225 **virtual triangle**: A virtual triangle is a leaf triangle added, if necessary, at the end of a partition to signal the end of the partition to the decoder.

3.226 **vop-bp**: An array of block-bps with the same accuracy significance in an fgs vop. There are three color components (Y, U, and V) in a vop-bp. Each color component in a vop-bp consists of all the block-bps of that color.

3.227 **VOP reordering**: The process of reordering the reconstructed VOPs when the decoding order is different from the display (or composition) order. VOP reordering occurs when B-VOPs are present in a bitstream. There is no VOP reordering when decoding low delay bitstreams.

3.228 **warping**: Processing applied to extract a sprite VOP from a static sprite or a reference VOP. It consists of a global spatial transformation driven by a few motion parameters (0,2,4,6,8), to recover luminance, chrominance and shape information.

3.229 **Y-vertex**: The vertex of a branching triangle opposite to the incident marching edge.

3.230 **zigzag scanning order**: A specific sequential ordering of the DCT coefficients from (approximately) the lowest spatial frequency to the highest.

4 Abbreviations and symbols

The mathematical operators used to describe this part of ISO/IEC 14496 are similar to those used in the C programming language. However, integer divisions with truncation and rounding are specifically defined. Numbering and counting loops generally begin from zero.

4.1 Arithmetic operators

+ Addition.

- Subtraction (as a binary operator) or negation (as a unary operator).

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

++ Increment. i.e. *x*++ is equivalent to *x* = *x* + 1

- - Decrement. i.e. *x*-- is equivalent to *x* = *x* - 1

$\left.\begin{matrix}\times\\ *\end{matrix}\right\}$ Multiplication.

^ Power.

/ Integer division with truncation of the result toward zero. For example, 7/4 and -7/-4 are truncated to 1 and -7/4 and 7/-4 are truncated to -1.

// Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example 3//2 is rounded to 2, and -3//2 is rounded to -2.

/// Integer division with sign dependent rounding to the nearest integer. Half-integer values when positive are rounded away from zero, and when negative are rounded towards zero. For example 3///2 is rounded to 2, and -3///2 is rounded to -1.

//// Integer division with truncation towards the negative infinity.

÷ Used to denote division in mathematical equations where no truncation or rounding is intended.

% Modulus operator. Defined only for positive numbers.

Sign( ) $\mathrm{Sign}(x) = \begin{cases} 1 & x >= 0 \\ -1 & x < 0 \end{cases}$

Abs( ) $\mathrm{Abs}(x) = \begin{cases} x & x >= 0 \\ -x & x < 0 \end{cases}$

$\sum_{i=a}^{i<b} f(i)$ The summation of the *f*(*i*) with *i* taking integral values from *a* up to, but not including *b*.

4.2 Logical operators

|| Logical OR.

&& Logical AND.

! Logical NOT.

4.3 Relational operators

> Greater than.

>= Greater than or equal to.

≥ Greater than or equal to.

< Less than.

<= Less than or equal to.

≤ Less than or equal to.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

== Equal to.

!= Not equal to.

max [, … ,] the maximum value in the argument list.

min [, … ,] the minimum value in the argument list.

4.4 Bitwise operators

& AND

| OR

>> Shift right with sign extension.

<< Shift left with zero fill.

4.5 Conditional operators

?: $$(condition ? a : b) = \begin{cases} a & \text{if } condition \text{ is true,} \\ b & \text{otherwise.} \end{cases}$$

4.6 Assignment

= Assignment operator.

4.7 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bitstream.

bslbf Bit string, left bit first, where “left” is the order in which bit strings are written in this part of ISO/IEC 14496. Bit strings are written as a string of 1s and 0s within single quote marks, e.g. ‘1000 0001’. Blanks within a bit string are for ease of reading and have no significance. For convenience large strings are occasionally written in hexadecimal, in this case conversion to a binary in the conventional manner will yield the value of the bit string. Thus the left most hexadecimal digit is first and in each hexadecimal digit the most significant of the four bits is first.

uimsbf Unsigned integer, most significant bit first.

simsbf Signed integer, in twos complement format, most significant (sign) bit first.

vlclbf Variable length code, left bit first, where “left” refers to the order in which the VLC codes are written. The byte order of multibyte words is most significant byte first.

4.8 Constants

Π 3,141 592 653 58…

e 2,718 281 828 45…

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5 Conventions

5.1 Method of describing bitstream syntax

The bitstream retrieved by the decoder is described in subclause 6.2. Each data item in the bitstream is in bold type. It is described by its name, its length in bits, and a mnemonic for its type and order of transmission.

The action caused by a decoded data element in a bitstream depends on the value of that data element and on data elements previously decoded. The decoding of the data elements and definition of the state variables used in their decoding are described in subclause 6.3. The following constructs are used to express the conditions when data elements are present, and are in normal type:

while (condition) {
data_element
. . .
}
	If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.

do { **data_element** . . . } while (condition)	The data element always occurs at least once. The data element is repeated until the condition is not true.
do { . . . continue . . . } while (condition)	Continues execution of the next repetition of the nearest while-do loop.

if (condition) {
data_element
. . .
} else {
data_element
. . .
}
	If the condition is true, then the first group of data elements occurs next in the data stream.
	
	If the condition is not true, then the second group of data elements occurs next in the data stream.

for (i = m; i < n; i++) {
data_element
. . .
}
	The group of data elements occurs (n-m) times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is set to m for the first occurrence, incremented by one for the second occurrence, and so forth.

/* comment . . . */	Explanatory comment that may be deleted entirely without in any way altering the syntax.

This syntax uses the ‘C-code’ convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is true and a variable or expression evaluating to a zero value is equivalent to a condition that is false. In many cases a literal string is used in a condition. For example;

if (video_object_layer_shape == “rectangular”) . . .

In such cases the literal string is that used to describe the value of the bitstream element in subclause 6.3. In this example, we see that “rectangular” is defined in a Table 6–16 to be represented by the two bit binary number ‘00’.

As noted, the group of data elements may contain nested conditional constructs. For compactness, the brackets { } are omitted when only one data element follows.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

data_element [n] data_element [n] is the n+1th element of an array of data.

data_element [m][n] data_element [m][n] is the m+1, n+1th element of a two-dimensional array of data.

data_element [l][m][n] data_element [l][m][n] is the l+1, m+1, n+1th element of a three-dimensional array of data.

While the syntax is expressed in procedural terms, it should not be assumed that subclause 6.2 implements a satisfactory decoding procedure. In particular, it defines a correct and error-free input bitstream. Actual decoders must include means to look for start codes in order to begin decoding correctly, and to identify errors, erasures or insertions while decoding. The methods to identify these situations, and the actions to be taken, are not standardised.

5.2 Definition of functions

Several utility functions for picture coding algorithm are defined as follows:

5.2.1 Definition of next_bits() function

The function next_bits() permits comparison of a bit string with the next bits to be decoded in the bitstream.

5.2.2 Definition of bytealigned() function

The function bytealigned () returns 1 if the current position is on a byte boundary, that is the next bit in the bitstream is the first bit in a byte. Otherwise it returns 0.

5.2.3 Definition of nextbits_bytealigned() function

The function nextbits_bytealigned() returns a bit string starting from the next byte aligned position. This permits comparison of a bit string with the next byte aligned bits to be decoded in the bitstream. If the current location in the bitstream is already byte aligned and the 8 bits following the current location are '01111111', the bits subsequent to these 8 bits are returned. The current location in the bitstream is not changed by this function.

5.2.4 Definition of next_start_code() function

The next_start_code() function removes any zero bit and a string of 0 to 7 '1' bits used for stuffing and locates the next start code.

next_start_code() {	**No. of bits**	**Mnemonic**
zero_bit	1	'0'
while (!bytealigned())		
one_bit	1	'1'
}		

5.2.5 Definition of next_resync_marker() function

The next_resync_marker() function removes any zero bit and a string of 0 to 7 '1' bits used for stuffing and locates the next resync marker; it thus performs similar operation as next_start_code() but for resync_marker.

next_resync_marker() {	**No. of bits**	**Mnemonic**
zero_bit	1	'0'
while (!bytealigned())		
one_bit	1	'1'
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.6 Definition of transparent_mb() function

The function transparent_mb() returns 1 if the current macroblock consists only of transparent pixels. Otherwise it returns 0.

5.2.7 Definition of transparent_block() function

The function transparent_block(j) returns 1 if the 8x8 with index j consists only of transparent pixels. Otherwise it returns 0. The index value for each block is defined in Figure 6-8.

5.2.8 Definition of byte_align_for_upstream() function

The function byte_align_for_upstream() removes a string of '1' used for stuffing from the upstream message. When the message is already byte aligned before the byte_align_for_upstream() function, additional byte stuffing is no longer allowed.

byte_align_for_upstream() {	**No. of bits**	**Mnemonic**
while (!byte_aligned())		
one_bit	1	'1'
}		

5.2.9 Definition of next_start_code_studio() function

The next_start_code_studio() function removes any zero bit and zero byte stuffing and locates the next start code.

next_start_code_studio() {	**No. of bits**	**Mnemonic**
while (!bytealigned())		
zero_bit	1	'0'
while (nextbits() != '0000 0000 0000 0000 0000 0001')		
zero_byte	8	'0000 0000'
}		

This function checks whether the current position is byte aligned. If it is not, zero stuffing bits are present. After that any number of zero stuffing bytes may be present before the start code. Therefore start codes are always byte aligned and may be preceded by any number of zero stuffing bits.

5.2.10 Definition of valid_stuffing_bits() function

If the current position in not on a byte boundary, the function valid_stuffing_bits() returns 1 if the remaining bits in the current byte are valid stuffing bits. If the current position is on a byte boundary, valid_stuffing_bits() returns 1 if the remaining bits in the next byte are valid stuffing bits. Otherwise, valid_stuffing_bits() returns 0.

5.2.11 Definition of start_of_bit_plane() function

The function start_of_bit_plane() returns 1 if the next bit in the bitstream is the first bit of the codes associated with a vop-bp. Otherwise it returns 0.

5.3 Reserved, forbidden and marker_bit

The terms "reserved" and "forbidden" are used in the description of some values of several fields in the coded bitstream.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The term “reserved” indicates that the value may be used in the future for ISO/IEC defined extensions.

The term “forbidden” indicates a value that shall never be used (usually in order to avoid emulation of start codes).

The term “marker_bit” indicates a one bit integer in which the value zero is forbidden (and it therefore shall have the value ‘1’). These marker bits are introduced at several points in the syntax to avoid start code emulation.

The term “zero_bit” indicates a one bit integer with the value zero.

5.4 Arithmetic precision

In order to reduce discrepancies between implementations of this part of ISO/IEC 14496, the following rules for arithmetic operations are specified.

(a) Where arithmetic precision is not specified, such as in the calculation of the IDCT, the precision shall be sufficient so that significant errors do not occur in the final integer values.

(b) Where ranges of values are given, the end points are included if a square bracket is present, and excluded if a round bracket is used. For example, [a , b) means from a to b, including a but excluding b.

6 Visual bitstream syntax and semantics

6.1 Structure of coded visual data

Coded visual data can be of several different types, such as video data, still texture data, 2D mesh data or facial animation parameter data.

Synthetic objects and their attribution are structured in a hierarchical manner to support both bitstream scalability and object scalability. ISO/IEC 14496-1:2001 of the specification provides the approach to spatial-temporal scene composition including normative 2D/3D scene graph nodes and their composition supported by Binary Interchange Format Specification. At this level, synthetic and natural object composition relies on ISO/IEC 14496-1:2001 with subsequent (non-normative) rendering performed by the application to generate specific pixel-oriented views of the models.

Coded video data consists of an ordered set of video bitstreams, called layers. If there is only one layer, the coded video data is called non-scalable video bitstream. If there are two layers or more, the coded video data is called a scalable hierarchy.

One of the layers is called base layer, and it can always be decoded independently. Other layers are called enhancement layers, and can only be decoded together with the lower layers (previous layers in the ordered set), starting with the base layer. The multiplexing of these layers is specified in ISO/IEC 14496-1:2001. The base layer of a scalable set of streams can be coded by other standards. The Enhancement layers shall conform to this part of ISO/IEC 14496. In general the visual bitstream can be thought of as a syntactic hierarchy in which syntactic structures contain one or more subordinate structures. In a typical application of FGS, the bitstream at the input of an FGS decoder is a truncated version of the bitstream at the output of an FGS encoder. It is likely that, at the end of each fgs vop before the next fgs_vop_start_code, only partial bits of the fgs vop are at the input of the decoder due to truncation of the fgs vop bitstream. Decoding of the truncated bitstream is not normative. An example of dealing with the truncated bitstream is described in Annex S. The FGS syntax description in this clause is for a complete bitstream without truncation.

Visual texture, referred to herein as still texture coding, is designed for maintaining high visual quality in the transmission and rendering of texture under widely varied viewing conditions typical of interaction with 2D/3D synthetic scenes. Still texture coding provides for a multi-layer representation of luminance, color and shape. This supports progressive transmission of the texture for image build-up as it is received by a terminal. Also supported is the downloading of the texture resolution hierarchy for construction of image pyramids used by 3D graphics APIs. Quality and SNR scalability are supported by the structure of still texture coding.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Coded mesh data consists of a single non-scalable bitstream. This bitstream defines the structure and motion of a 2D mesh object. Texture that is to be mapped onto the mesh geometry is coded separately.

Coded FBA parameter data consists of one non-scaleable bitstream. It defines the animation of the face and body model of the decoder. Face and body animation data is structured as standard formats for downloadable models and their animation controls, and a single layer of compressed face and body animation parameters used for remote manipulation of the face and body model. The face is a node in a scene graph that includes face geometry ready for rendering. The body is also a node in a scene graph that includes body geometry ready for rendering. The face node belongs to the scene graph that is defined by the body node.The shape, texture and expressions of the face are controlled by the bitstream containing instances of Facial Definition Parameter (FDP) sets and/or Facial Animation Parameter (FAP) sets. The body is also controlled by instances of Body Definition Parameter (BDP) sets and/or Body Animation Parameter (BAP) sets.Upon initial or baseline construction, the FBA object contains a generic face with a neutral expression and a generic body with default position. This object can receive FAPs and BAPs from the bitstream and be subsequently rendered to produce animation of the face and the body. If FDPs and BDPs are transmitted, the generic face and body are transformed into a particular face and body of specific shape and appearance. A downloaded face model via FDPs is a scene graph for insertion in the face node, and the downloaded body model via BDPs is a scene graph for insertion of the body node.

6.1.1 Visual object sequence

Visual object sequence is the highest syntactic structure of the coded visual bitstream.

A visual object sequence commences with a visual_object_sequence_start_code which is followed by one or more visual objects coded concurrently. The visual object sequence is terminated by a visual_object_sequence_end_code.

At various points in the visual object sequence, a repeat visual_object_sequence_start_code can be inserted for coded video data. In that case, the repeat visual_object_sequence_start_code shall follow a particular VOP. When profile_and_level_indication indicates a Studio Profile, StudioVisualObject() shall follow it.

6.1.2 Visual object

A visual object commences with a visual_object_start_code, is followed by profile and level identification, and a visual object id, and is followed by a video object, a still texture object, a mesh object, or an FBA object. For Studio Profiles particularly, only video object type is supported.

6.1.3 Video object

A video object commences with a video_object_start_code, and is followed by one or more video object layers. For Studio Profiles particularly, a video object layer commences with video_object_layer_start_code which may optionally be followed by Group_of_StudioVideoObjectPlane() and then by one or more coded VOPs. The order of the coded frames in the coded bitstream is the order in which the decoder processes them, which is not necessarily the display order.

6.1.3.1 Progressive and interlaced sequences

This part of ISO/IEC 14496 deals with coding of both progressive and interlaced sequences.

The sequence, at the output of the decoding process, consists of a series of reconstructed VOPs separated in time and are readied for display via the compositor.

For Studio Profiles particularly, the output of the decoding process for interlaced sequences consists of a series of reconstructed fields that are separated in time by a field period. The two fields of a frame may be coded separately (field-VOPs). Alternatively the two fields may be coded together as a frame (frame-VOPs). Both frame VOPs and field VOPs may be used in a single video sequence.

In progressive sequences each VOP in the sequence shall be a frame VOP. The sequence, at the output of the decoding process, consists of a series of reconstructed frames that are separated in time by a frame period.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.3.2 Frame

A frame consists of three rectangular matrices of integers; a luminance matrix (Y), and two chrominance matrices (Cb and Cr).

The relationship between these Y, Cb and Cr components and the primary (analogue) Red, Green and Blue Signals (E'_R, E'_G and E'_B), the chromaticity of these primaries and the transfer characteristics of the source frame may be specified in the bitstream (or specified by some other means). This information does not affect the decoding process.

For Studio Profiles particularly, the three rectangular matrices can be the primary RGB colour matrices.

6.1.3.3 Field

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame.

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices. The bottom field is the other one.

6.1.3.4 VOP

A reconstructed VOP is obtained by decoding a coded VOP. A coded VOP may have been derived from either a progressive or interlaced frame.

For Studio Profiles particularly, coded VOP may have been derived from a progressive or interlaced frame or an interlaced field. A reconstructed VOP is either a reconstructed frame (when decoding a frame VOP), or one field of a reconstructed frame (when decoding a field VOP).

An I-frame VOP or a pair of field VOPs, where the first field VOP is an I-picture and the second field VOP is an I-VOP or a P-VOP, is called a coded I-frame.

A P-frame VOP or a pair of P-field VOPs is called a coded P-frame.

A B-frame VOP or a pair of B-field VOPs is called a coded B-frame.

A coded I-frame, a coded P-frame or a coded B-frame is called a coded frame

6.1.3.4.1 Field VOPs

If field VOPs are used, then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field VOPs that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first VOP of the coded frame is a P-field VOP, then the second VOP of the coded frame shall also be a P-field VOP. Similarly when the first VOP of the coded frame is a B-field VOP the second VOP of the coded frame shall also be a B-field VOP.

When the first VOP of the coded frame is a I-field VOP, then the second VOP of the frame shall be either an I-field VOP or a P-field VOP. If the second VOP is a P-field VOP, then certain restrictions apply (see 7.16.7.4.5).

6.1.3.4.2 Frame VOPs

When coding interlaced sequences using frame VOPs, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-VOP.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.3.5 VOP types

There are four types of VOPs that use different coding methods:

1. An Intra-coded (I) VOP is coded using information only from itself.

2. A Predictive-coded (P) VOP is a VOP which is coded using motion compensated prediction from a past reference VOP.

3. A Bidirectionally predictive-coded (B) VOP is a VOP which is coded using motion compensated prediction from a past and/or future reference VOP(s).

4. A sprite (S) VOP is a VOP for a sprite object or a VOP which is coded using prediction based on global motion compensation from a past reference VOP.

6.1.3.6 I-VOPs and group of VOPs

I-VOPs are intended to assist random access into the sequence. Applications requiring random access, fast-forward playback, or fast reverse playback may use I-VOPs relatively frequently.

I-VOPs may also be used at scene cuts or other cases where motion compensation is ineffective.

Group of VOP (GOV) header is an optional header that can be used immediately before a coded I-VOP to indicate to the decoder:

1) the modulo part (i.e. the full second units) of the time base for the next VOP to be displayed after having decoded a GOV header. For Studio Profiles particularly, SMPTE 12M time code information that is not used by the decoding process.

2) if the first consecutive B-VOPs immediately following the coded I-VOP can be reconstructed properly in the case of a random access.

In a non scalable bitstream or the base layer of a scalable bitstream, the first coded VOP following a GOV header shall be a coded I-VOP.

6.1.3.7 Format

6.1.3.7.1 4:2:0 Format

In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in both horizontal and vertical dimensions. The Y-matrix shall have an even number of lines and samples.

If the matrices represent RGB colour primary matrices, this 4:2:0 format shall not be applied.

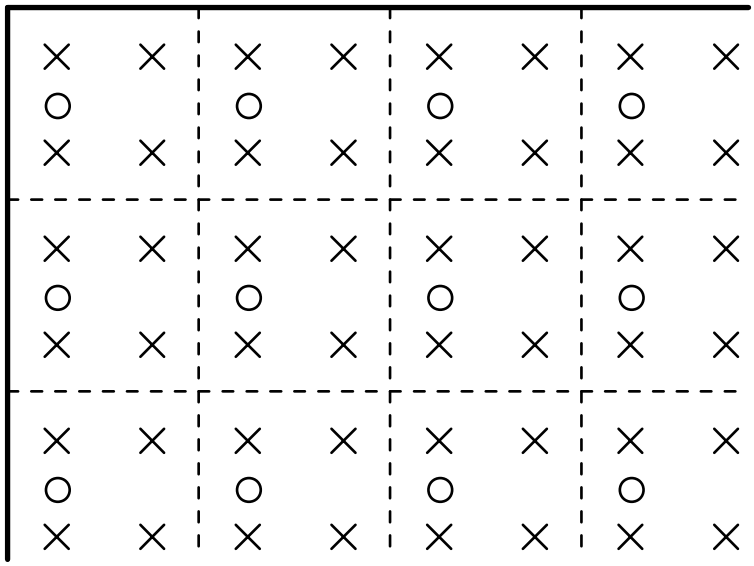
NOTE — When interlaced frames are coded as rectangular field VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by four.

The luminance and chrominance samples are positioned as shown in Figure 6-1.The two variations in the vertical and temporal positioning of the samples for interlaced VOPs are shown in Figure 6-2 and Figure 6-3.

Figure 6-4 shows the vertical and temporal positioning of the samples in a progressive frame.

In each field of an interlaced frame, the chrominance samples do not lie (vertically) mid way between the luminance samples of the field. This is so that the spatial location of the chrominance samples in the frame is the same whether the frame is represented as a single frame-VOP or two field-VOPs.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



× Represent luminance samples

○ Represent chrominance samples

Figure 6-1 — The position of luminance and chrominance samples in 4:2:0 data

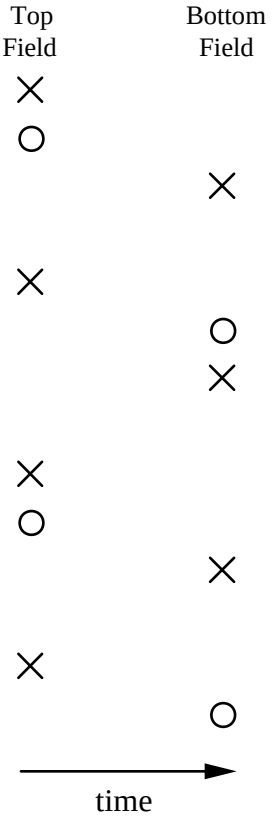


Figure 6-2 — Vertical and temporal positions of samples in an interlaced frame with top_field_first=1

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

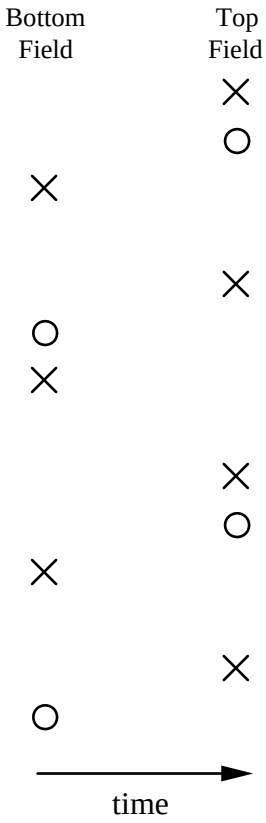


Figure 6-3 — Vertical and temporal position of samples in an interlaced frame with top_field_first=0

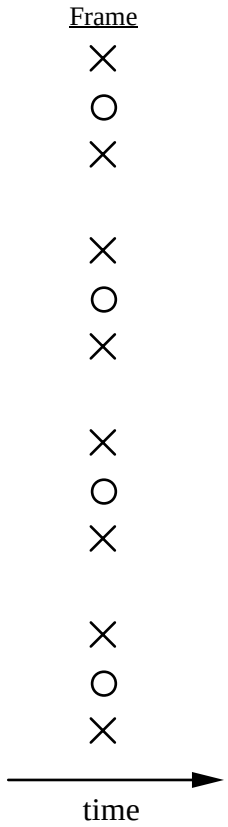


Figure 6-4 — Vertical and temporal positions of samples in a progressive frame

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The binary alpha plane for each VOP is represented by means of a bounding rectangle as described in Clause F.2, and it has always the same number of lines and pixels per line as the luminance plane of the VOP bounding rectangle. The positions between the luminance and chrominance pixels of the bounding rectangle are defined in this clause according to the 4:2:0 format. For the progressive case, each 2x2 block of luminance pixels in the bounding rectangle associates to one chrominance pixel. For the interlaced case, each 2x2 block of luminance pixels of the same field in the bounding rectangle associates to one chrominance pixel of that field.

In order to perform the padding process on the two chrominance planes, it is necessary to generate a binary alpha plane which has the same number of lines and pixels per line as the chrominance planes. Therefore, when non-scalable shape coding is used, this binary alpha plane associated with the chrominance planes is created from the binary alpha plane associated with the luminance plane by the subsampling process defined below:

For each 2x2 block of the binary alpha plane associated with the luminance plane of the bounding rectangle (of the same frame for the progressive and of the same field for the interlaced case), the associated pixel value of the binary alpha plane associated with the chrominance planes is set to 255 if any pixel of said 2x2 block of the binary alpha plane associated with the luminance plane equals 255.

6.1.3.7.2 4:2:2 Format

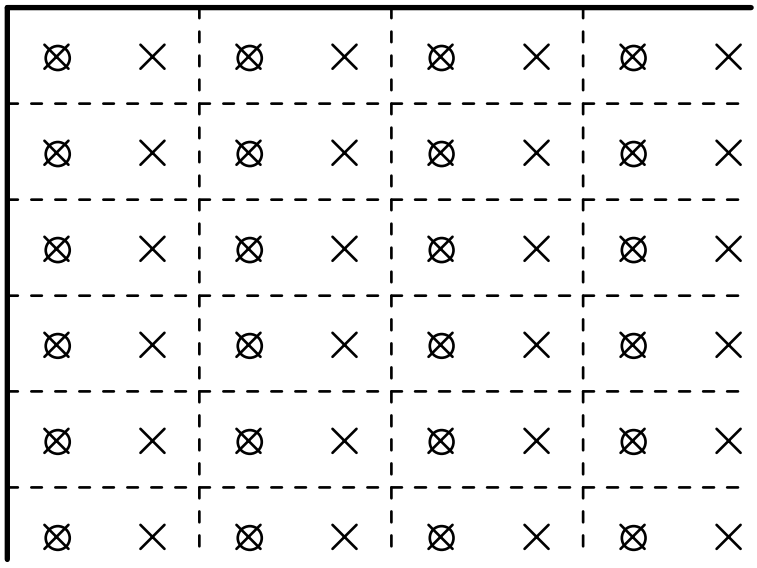
In this format the Cb and Cr matrices shall be one half the size of the Y-matrix in the horizontal dimension and the same size as the Y-matrix in the vertical dimension. The Y-matrix shall have an even number of samples.

If the matrices represent RGB colour primar matrices, this 4:2:2 format shall not be applied.

NOTE — When interlaced frames are coded as rectangular field VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figure 6-5.

In order to clarify the organisation, Figure 6-6 shows the (vertical) positioning of the samples when the frame is separated into two fields.



× Represent luminance samples

○ Represent chrominance samples

Figure 6-5 — The position of luminance and chrominance samples. 4:2:2 data.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

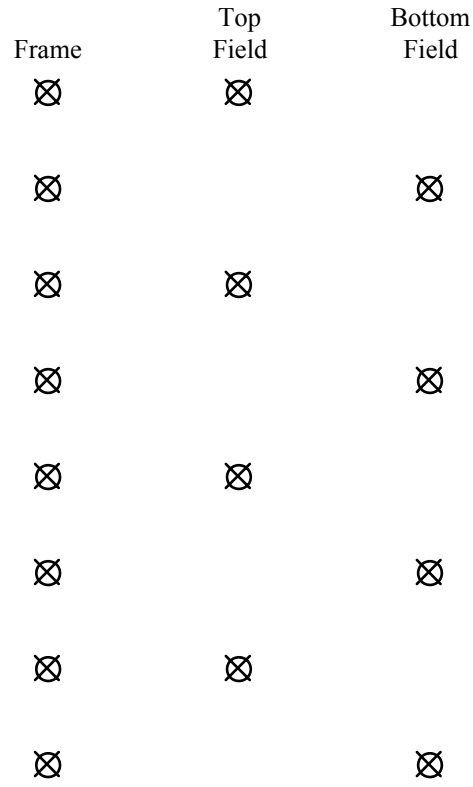


Figure 6-6 — Vertical positions of samples with 4:2:2 and 4:4:4 data

6.1.3.7.3 4:4:4 Format

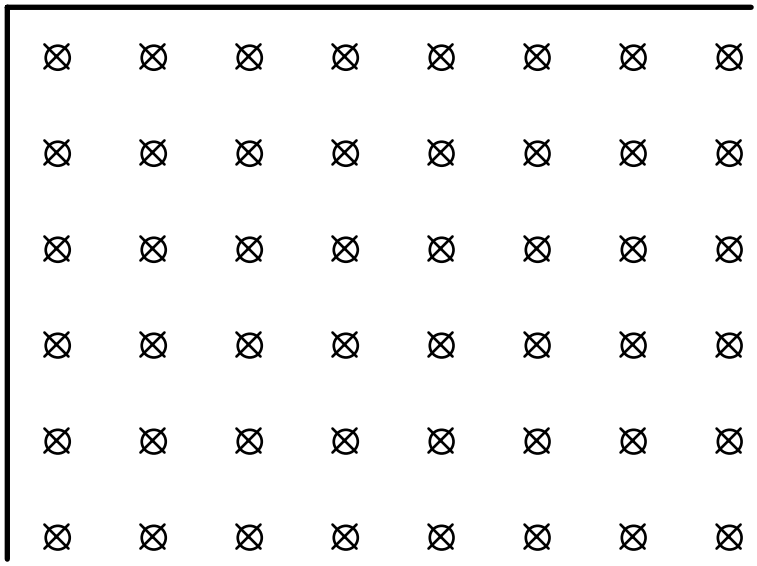
In this format the Cb and Cr matrices shall be the same size as the Y-matrix in the horizontal and the vertical dimensions.

If the matrices are treated as RGB colour primary matrices, the matrices shall follow this format.

NOTE — When interlaced frames are coded as field rectangular VOPs, the VOP reconstructed from each of these field VOPs shall have a Y-matrix with half the number of lines of the corresponding frame. Thus the total number of lines in the Y-matrix of an entire frame shall be divisible by two.

The luminance and chrominance samples are positioned as shown in Figure 6-6 and Figure 6-7.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



× Represent luminance samples

○ Represent chrominance samples

Figure 6-7 — The position of luminance and chrominance samples. 4:4:4 data.

6.1.3.8 VOP reordering

When a video object layer contains coded B-VOPs, the number of consecutive coded B-VOP is variable and unbounded. The first coded VOP shall not be a B-VOP.

A video object layer may contain no coded P-VOPs or no coded S(GMC)-VOPs. A video object layer may also contain no coded I-VOPs in which case some care is required at the start of the video object layer and within the video object layer to effect both random access and error recovery.

The order of the coded VOPs in the bitstream, also called decoding order, is the order in which a decoder reconstructs them. The order of the reconstructed VOPs at the output of the decoding process, also called the display order, is not always the same as the decoding order and this subclause defines the rules of VOP reordering that shall happen within the decoding process.

When the video object layer contains no coded B-VOPs, the decoding order is the same as the display order.

When B-VOPs are present in the video object layer re-ordering is performed according to the following rules:

If the current VOP in decoding order is a B-VOP the output VOP is the VOP reconstructed from that B-VOP.

If the current VOP in decoding order is a I-VOP, P-VOP, or S(GMC)-VOPs the output VOP is the VOP reconstructed from the previous I-VOP or P-VOP, or S(GMC)-VOP if one exists. If none exists, at the start of the video object layer, no VOP is output.

The following is an example of VOPs taken from the beginning of a video object layer. In this example there are two coded B-VOPs between successive coded P-VOPs and also two coded B-VOPs between successive coded I- and P-VOPs. VOP ‘1I’ is used to form a prediction for VOP ‘4P’. VOPs ‘4P’ and ‘1I’ are both used to form predictions for VOPs ‘2B’ and ‘3B’. Therefore the order of coded VOPs in the coded sequence shall be ‘1I’, ‘4P’, ‘2B’, ‘3B’. However, the decoder shall display them in the order ‘1I’, ‘2B’, ‘3B’, ‘4P’.

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

At the encoder input,

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	I	B	B	P

At the encoder output, in the coded bitstream, and at the decoder input,

1	4	2	3	7	5	6	10	8	9	13	11	12
I	P	B	B	P	B	B	I	B	B	P	B	B

At the decoder output,

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	I	B	B	P

In terms of VOP reordering, an S(GMC)-VOP can be regarded as a P-VOP.

6.1.3.9 Macroblock

A macroblock contains a section of the luminance component and the spatially corresponding chrominance components. The term macroblock can either refer to source and decoded data or to the corresponding coded data elements. A skipped macroblock is one for which no information is transmitted. There are three chrominance formats for a macroblock, namely, 4:2:0, 4:2:2 and 4:4:4 formats. The order of blocks in a macroblock shall be different for each different chrominance format and are illustrated below:

A 4:2:0 Macroblock consists of 6 blocks. This structure holds 4 Y, 1 Cb and 1 Cr Blocks and the block order is depicted in Figure 6-8.

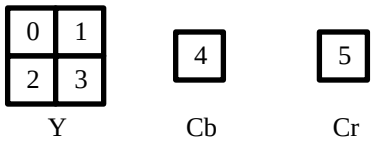


Figure 6-8 — 4:2:0 Macroblock structure

A 4:2:2 Macroblock consists of 8 blocks. This structure holds 4 Y, 2 Cb and 2 Cr Blocks and the block order is depicted in Figure 6-9.

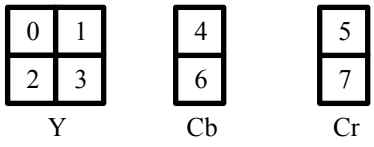


Figure 6-9 — 4:2:2 Macroblock structure

A 4:4:4 Macroblock consists of 12 blocks. This structure holds 4 Y, 4 Cb and 4 Cr (or 4 G, 4 B and 4 R) Blocks and the block order is depicted in Figure 6-10.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

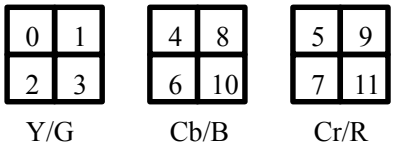


Figure 6-10 — 4:4:4 Macroblock structure

The organisation of VOPs into macroblocks is as follows.

For the case of a progressive VOP (when the interlaced flagin the VOL header is set to “0”), the organisation of lines of luminance VOP into macroblocks is called frame organization and is illustrated in Figure 6-11. In this case, frame DCT coding is employed.

For the case of interlaced VOP (when the interlaced flag is set to “1”), the organisation of lines of luminance VOP into macroblocks can be either frame organization or field organization and thus both frame and field DCT coding may be used in the VOP.

- In the case of frame DCT coding, each luminance block shall be composed of lines from two fields alternately. This is illustrated in Figure 6-11.

- In the case of field DCT coding, each luminance block shall be composed of lines from only one of the two fields. This is illustrated in Figure 6-12.

In the case of chrominance blocks the structure depends upon the chrominance format that is being used. In the case of 4:2:2 and 4:4:4 formats (where there are two blocks in the vertical dimension of the macroblock) the chrominance blocks are treated in exactly the same manner as the luminance blocks. However, in the 4:2:0 format the chrominance blocks shall always be organised in frame structure for the purposes of DCT coding. It should however be noted that field based predictions may be made for these blocks which will, in the general case, require that predictions for 8x4 regions (after half-sample filtering) must be made.

In field pictures, each picture only contains lines from one of the fields. In this case each block consists of lines taken from successive lines in the picture as illustrated by Figure 6-11.

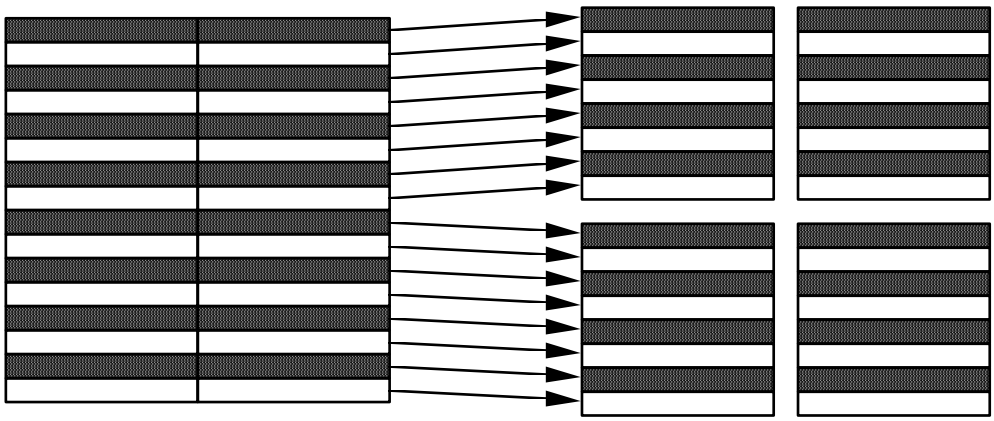


Figure 6-11 — Luminance macroblock structure in frame DCT coding

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

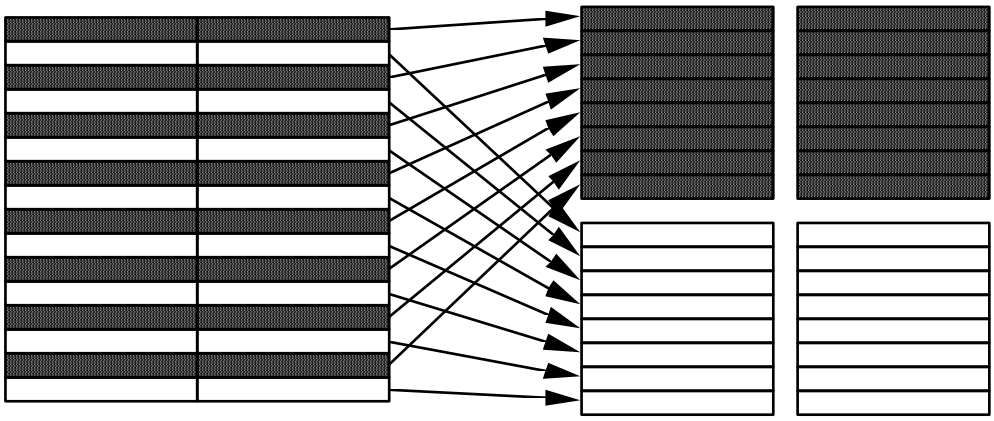


Figure 6-12 — Luminance macroblock structure in field DCT coding

6.1.3.10 Block

The term **block** can refer either to source and reconstructed data or to the DCT coefficients or to the corresponding coded data elements.

When the block refers to source and reconstructed data it refers to an orthogonal section of a luminance or chrominance component with the same number of lines and samples. There are 8 lines and 8 samples/line in the block.

6.1.3.11 Field

A field consists of every other line of samples in the three rectangular matrices of integers representing a frame.

A frame is the union of a top field and a bottom field. The top field is the field that contains the top-most line of each of the three matrices. The bottom field is the other one.

Only when profile_and_level_indication indicates the studio profile, a coded VOP may be a frame VOP or a field VOP. A reconstructed VOP is either a reconstructed frame (when decoding a frame VOP), or one field of a reconstructed frame (when decoding a field VOP).

6.1.3.11.1 Field VOPs

If field VOPs are used then they shall occur in pairs (one top field followed by one bottom field, or one bottom field followed by one top field) and together constitute a coded frame. The two field VOPs that comprise a coded frame shall be encoded in the bitstream in the order in which they shall occur at the output of the decoding process.

When the first VOP of the coded frame is a P-field VOP, then the second VOP of the coded frame shall also be a P- field VOP. Similarly when the first VOP of the coded frame is a B-field VOP the second VOP of the coded frame shall also be a B-field VOP.

When the first VOP of the coded frame is a I-field VOP, then the second VOP of the frame shall be either an I-field VOP or a P-field VOP. If the second VOP is a P-field VOP then certain restrictions apply.

6.1.3.11.2 Frame VOPs

When coding interlaced sequences using frame VOPs, the two fields of the frame shall be interleaved with one another and then the entire frame is coded as a single frame-VOP.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.3.12 Slice

A slice is a series of an arbitrary number of consecutive macroblocks. The first and last macroblocks of a slice shall not be skipped macroblocks. Every slice shall contain at least one macroblock. Slices shall not overlap. The position of slices may change from picture to picture.

The first and last macroblock of a slice shall be in the same horizontal row of macroblocks.

Slices shall occur in the bitstream in the order in which they are encountered, starting at the upper-left of the picture and proceeding by raster-scan order from left to right and top to bottom (illustrated in the Figures of this clause as alphabetical order).

6.1.3.12.1 The general slice structure

In the most general case it is not necessary for the slices to cover the entire picture. Figure 6-13 shows this case. Those areas that are not enclosed in a slice are not encoded and no information is encoded for such areas (in the specific picture).

If the slices do not cover the entire picture then it is a requirement that if the picture is subsequently used to form predictions then predictions shall only be made from those regions of the picture that were enclosed in slices. It is the responsibility of the encoder to ensure this.

This specification does not define what action a decoder shall take in the regions between the slices.

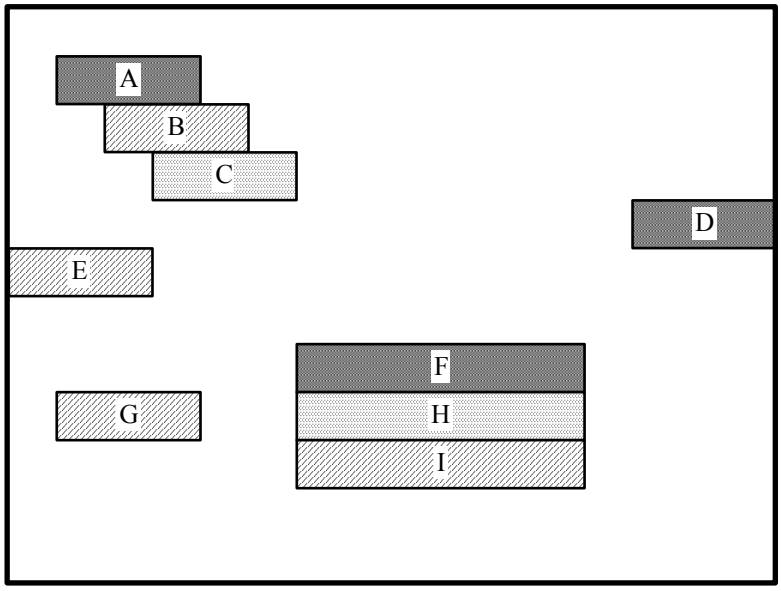


Figure 6-13 — The most general slice structure

6.1.3.12.2 Restricted slice structure

In certain defined levels of defined profiles a restricted slice structure illustrated in Figure 6-14 shall be used. In this case every macroblock in the picture shall be enclosed in a slice.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

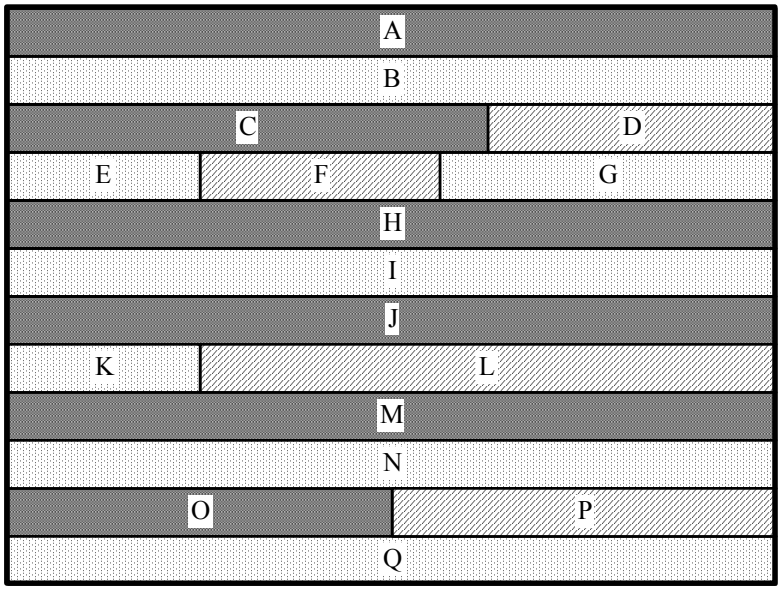


Figure 6-14 — Restricted slice structure

Where a defined level of a defined profile requires that the slice structure obeys the restrictions detailed in this clause, the term “restricted slice structure” may be used.

6.1.4 Mesh object

A 2D triangular *mesh* refers to a tessellation of a 2D visual object plane into triangular patches. The vertices of the triangular patches are called *node points*. The straight-line segments between node points are called *edges*. Two triangles are *adjacent* if they share a common edge.

A *dynamic* 2D mesh consists of a temporal sequence of 2D triangular meshes, where each mesh has the same topology, but node point locations may differ from one mesh to the next. Thus, a dynamic 2D mesh can be specified by the geometry of the initial 2D mesh and motion vectors at the node points for subsequent meshes, where each motion vector points from a node point of the previous mesh in the sequence to the corresponding node point of the current mesh. The dynamic 2D mesh can be used to create 2D animations by mapping texture from e.g. a video object plane onto successive 2D meshes.

A 2D dynamic mesh with *implicit structure* refers to a 2D dynamic mesh of which the initial mesh has either *uniform* or *Delaunay* topology. In both cases, the topology of the initial mesh does not have to be coded (since it is implicitly defined), only the node point locations of the initial mesh have to be coded. Note that in both the uniform and Delaunay case, the mesh is restricted to be *simple*, i.e. it consists of a single connected component without any holes, topologically equivalent to a disk.

A *mesh object* represents the geometry and motion of a 2D triangular mesh. A mesh object consists of one or more *mesh object planes*, each corresponding to a 2D triangular mesh at a certain time instance. An example of a mesh object is shown in the Figure 6-15.

A sequence of mesh object planes represents the piece-wise deformations to be applied to a video object plane or still texture object to create a synthetic animated video object. Triangular patches of a video object plane are to be warped according to the motion of corresponding triangular mesh elements. The motion of mesh elements is specified by the temporal displacements of the mesh node points.

The syntax and semantics of the mesh object pertains to the mesh geometry and mesh motion only; the video object to be used in an animation is coded separately. The warping or texture mapping applied to render visual object planes is handled in the context of scene composition. Furthermore, the syntax does not allow explicit encoding of other mesh properties such as colors or texture coordinates.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

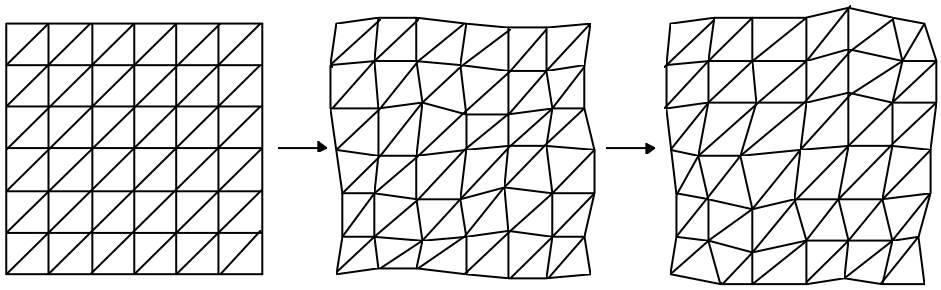


Figure 6-15 — Mesh object with uniform triangular geometry

6.1.4.1 Mesh object plane

There are two types of mesh object planes that use different coding methods.

An *intra-coded* mesh object plane codes the geometry of a single 2D mesh. An intra-coded mesh is either of uniform or Delaunay type. In the case of a mesh of uniform type, the mesh geometry is coded by a small set of parameters. In the case of a mesh of Delaunay type, the mesh geometry is coded by the locations of the node points and boundary edge segments. The triangular mesh structure is specified implicitly by the coded information.

A *predictive-coded* mesh object plane codes a 2D mesh using temporal prediction from a past reference mesh object plane. The triangular structure of a predictive-coded mesh is identical to the structure of the reference mesh used for prediction; however, the locations of node points may change. The displacements of node points represent the motion of the mesh and are coded by specifying the motion vectors of node points from the reference mesh towards the predictive-coded mesh.

The locations of mesh node points correspond to locations in a video object or still texture object. Mesh node point locations and motion vectors are represented and coded with half pixel accuracy.

6.1.5 FBA object

Conceptually the FBA object consists of a collection of nodes in a scene graph which are animated by the FBA object bitstream. The shape, texture and expressions of the face are controlled by the bitstream containing instances of Facial Definition Parameter (FDP) sets and/or Facial Animation Parameter (FAP) sets. Upon construction, the FBA object contains a generic face with a neutral expression. This face can already be rendered. It is also immediately capable of receiving the FAPs from the bitstream, which will produce animation of the face: expressions, speech etc. If FDPs are received, they are used to transform the generic face into a particular face determined by its shape and (optionally) texture. Optionally, a complete face model can be downloaded via the FDP set as a scene graph for insertion in the face node.

The FDP and FAP sets are designed to allow the definition of a facial shape and texture, as well as animation of faces reproducing expressions, emotions and speech pronunciation. The FAPs, if correctly interpreted, will produce reasonably similar high level results in terms of expression and speech pronunciation on different facial models, without the need to initialize or calibrate the model. The FDPs allow the definition of a precise facial shape and texture in the setup phase. If the FDPs are used in the setup phase, it is also possible to produce more precisely the movements of particular facial features. Using a phoneme/bookmark to FAP conversion it is possible to control facial models accepting FAPs via TTS systems. The translation from phonemes to FAPs is not standardized. It is assumed that every decoder has a default face model with default parameters. Therefore, the setup stage is not necessary to create face animation. The setup stage is used to customize the face at the decoder.

Upon construction, the Body model contains a generic virtual human or human-like body with the default posture. This body can already be rendered. It is also immediately capable of receiving the BAPs from the bitstream, which will produce animation of the body. If BDPs are received, they are used to transform the decoder's generic body into a particular body determined by the parameter contents. Any component can be null. A null component is replaced by the corresponding default component when the body is rendered. Similar to the face, the BAPs can be transmitted also without first downloading BDPs, in which case the decoder animates its local model.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

No assumption is made and no limitation is imposed on the range of defined mobilities for humanoid animation. In other words the human body model should be capable of supporting various applications, from realistic simulation of human motions to network games using simple human-like models.

6.1.5.1 Structure of the face and body object bitstream

A face and body object is formed by a temporal sequence of face and body object planes. This is depicted as follows in Figure 6-16.

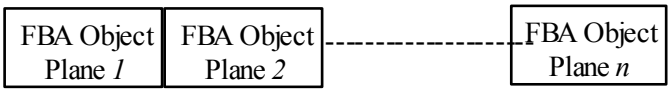
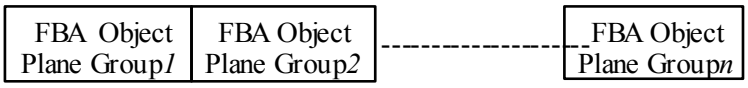


Figure 6-16 — Structure of the FBA object bitstream

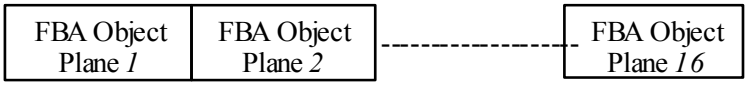
An FBA object represents a node in an ISO/IEC 14496 scene graph. An ISO/IEC 14496 scene is understood as a composition of Audio-Visual objects according to some spatial and temporal relationships. The scene graph is the hierarchical representation of the ISO/IEC 14496 scene structure (see ISO/IEC 14496-1:2001).

Alternatively, an FBA object can be formed by a temporal sequence of FBA object plane groups (called segments for simplicity), where each FBA object plane group itself is composed of a temporal sequence of 16 FBA object planes, as depicted in the following:

FBA **object:**



FBA **object plane group:**



When the alternative FBA object bitstream structure is employed, the bitstream is decoded by DCT-based FBA object decoding as described in subclause 7.12.2. Otherwise, the bitstream is decoded by the frame-based FBA object decoding. Refer to Table C-1 for a specification of default minimum and maximum values for each FAP and BAP.

6.1.5.2 Facial animation parameter set

The FAPs are based on the study of minimal facial actions and are closely related to muscle actions. They represent a complete set of basic facial actions, and therefore allow the representation of most natural facial expressions. Exaggerated values permit the definition of actions that are normally not possible for humans, but could be desirable for cartoon-like characters.

The FAP set contains two high level parameters visemes and expressions. A viseme is a visual correlate to a phoneme. The viseme parameter allows viseme rendering (without having to express them in terms of other parameters) and enhances the result of other parameters, insuring the correct rendering of visemes. Only static visemes which are clearly distinguished are included in the standard set. Additional visemes may be added in future extensions of the standard. Similarly, the expression parameter allows definition of high level facial expressions. The facial expression parameter values are defined by textual descriptions. To facilitate facial animation, FAPs that can be used together to represent natural expression are grouped together in FAP groups, and can be indirectly addressed by using an expression parameter. The expression parameter allows for a very efficient means of animating faces. In Annex C, a list of the FAPs is given, together with the FAP grouping, and the definitions of the facial expressions.

--`,,-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

6.1.5.3 Facial animation parameter units

All the parameters involving translational movement are expressed in terms of the *Facial Animation Parameter Units (FAPU)*. These units are defined in order to allow interpretation of the FAPs on any facial model in a consistent way, producing reasonable results in terms of expression and speech pronunciation. They correspond to fractions of distances between some key facial features and are defined in terms of distances between feature points. The fractional units used are chosen to allow enough precision. Annex C contains the list of the FAPs and the list of the FDP feature points. For each FAP the list contains the name, a short description, definition of the measurement units, whether the parameter is unidirectional (can have only positive values) or bi-directional, definition of the direction of movement for positive values, group number (for coding of selected groups), FDP subgroup number (Annex C) and quantisation step size. FAPs act on FDP feature points in the indicated subgroups. The measurement units are shown in Table 6-1, where the notation 3.1.y represents the y coordinate of the feature point 3.1; also refer to Figure 6-17.

Table 6-1 -- Facial Animation Parameter Units

Description		FAPU Value
IRISD0 = 3.1.y – 3.3.y = 3.2.y – 3.4.y	Iris diameter (by definition it is equal to the distance between upper ad lower eyelid) in neutral face	IRISD = IRISD0 / 1024
ES0 = 3.5.x – 3.6.x	Eye separation	ES = ES0 / 1024
ENS0 = 3.5.y – 9.15.y	Eye - nose separation	ENS = ENS0 / 1024
MNS0 = 9.15.y – 2.2.y	Mouth - nose separation	MNS = MNS0 / 1024
MW0 = 8.3.x – 8.4.x	Mouth width	MW = MW0 / 1024
AU	Angle Unit	10^{-5} rad

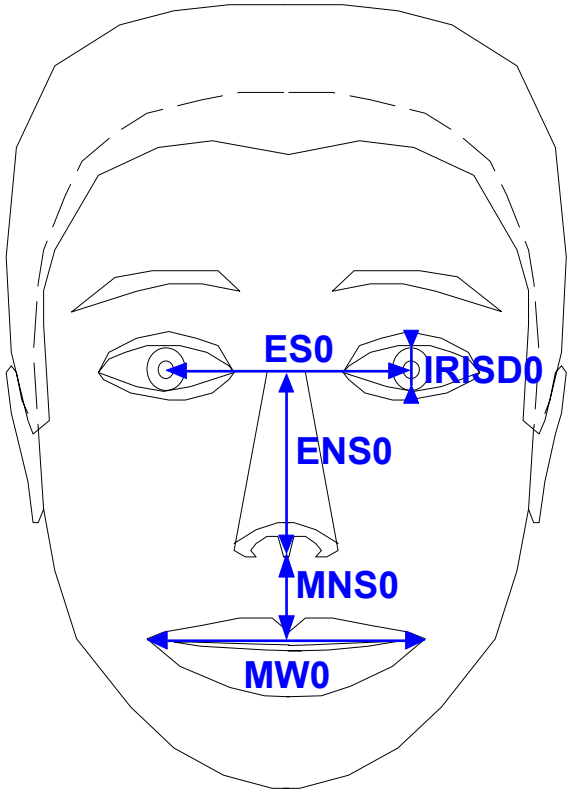


Figure 6-17 — The Facial Animation Parameter Units

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.5.4 Description of a neutral face

At the beginning of a sequence, the face is supposed to be in a neutral position. Zero values of the FAPs correspond to a neutral face. All FAPs are expressed as displacements from the positions defined in the neutral face. The neutral face is defined as follows:

- the coordinate system is right-handed; head axes are parallel to the world axes
- gaze is in direction of Z axis
- all face muscles are relaxed
- eyelids are tangent to the iris
- the pupil is one third of IRISD0
- lips are in contact; the line of the lips is horizontal and at the same height of lip corners
- the mouth is closed and the upper teeth touch the lower ones
- the tongue is flat, horizontal with the tip of tongue touching the boundary between upper and lower teeth (feature point 6.1 touching 9.11 in Annex C)

6.1.5.5 Facial definition parameter set

The FDPs are used to customize the proprietary face model of the decoder to a particular face or to download a face model along with the information about how to animate it. The definition and description of FDP fields is given in Annex C. The FDPs are normally transmitted once per session, followed by a stream of compressed FAPs. However, if the decoder does not receive the FDPs, the use of FAPUs ensures that it can still interpret the FAP stream. This insures minimal operation in broadcast or teleconferencing applications. The FDP set is specified in BIFS syntax (see ISO/IEC 14496-1:2001). The FDP node defines the face model to be used at the receiver. Two options are supported:

- calibration information is downloaded so that the proprietary face of the receiver can be configured using facial feature points and optionally a 3D mesh or texture.
- a face model is downloaded with the animation definition of the Facial Animation Parameters. This face model replace the proprietary face model in the receiver.

6.1.5.6 Body animation parameter set

BAP parameters comprise joint angles connecting different body parts. These include: toe, ankle, knee, hip, spine (C1-C7, T1-T12, L1-L5), shoulder, clavicle, elbow, wrist, and the hand fingers. The detailed joint list, with the rotation normals, are given in the following subclause. The rotation angles are assumed to be positive in the counterclockwise rotation direction with respect to the rotation normal. The rotation angles are defined as zero in the default posture, as defined below.

Note that the normals of rotation move with the body, and they are fixed with respect to the parent body part. That is to say, the axes of rotation are not aligned with the body or world coordinate system, but move with the body parts.

The hands are capable of performing complicated motions and are included in the body hierarchy. There are totally 29 degrees of freedom on each hand, assuming that the hand has a standard structure with five fingers.

The unit of rotations (BAPU) is defined as PI/10E-5 radians. The unit of translation BAPs (BAPs HumanoidRoot_tr_vertical, HumanoidRoot_tr_lateral, HumanoidRoot_tr_frontal) is defined in millimeters.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.5.7 Description of the default posture of the body

The default posture is defined by standing posture. This posture is defined as follows: the feet should point to the front direction, the two arms should be placed on the side of the body with the palm of the hands facing inward. This posture also implies that all BAPs have default values as 0.

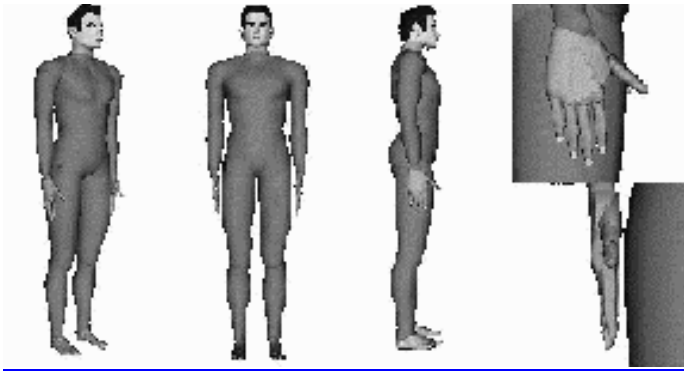


Figure 6-18

6.1.5.8 Body Definition Parameter Set

The BDPs are used to customize the proprietary body model of the decoder to a particular body or to download a body model along with the information about how to animate it. The definition and description of BDP fields is given in Annex C. The BDPs are normally transmitted once per session, followed by a stream of compressed BAPs. However, if the decoder does not receive the BDPs, model-independence of BAPs ensures that it can still interpret the BAP stream. This insures minimal operation in broadcast or teleconferencing applications. The BDP set is specified in BIFS syntax (see ISO/IEC 14496-1:2001). The BDP node defines the body model to be used at the receiver.

6.1.6 3D Mesh Object

The compressed bitstream for a 3D mesh is composed of a header data block with global information, followed by a sequence of connected component data blocks, each one associated with one connected component of the 3D mesh.

3D Mesh Header	CC Data #1	...	CC Data #nCC

If a 3D mesh is coded in error resilience mode, connected component data blocks are grouped or divided into partitions.

Partition #1	Partition #2	...	Partition #nPT

Additionally, if the 3D mesh is represented in hierarchical mode, the last connected component data block is followed by one or more refinement step data blocks, each one of them representing a Forest Split operation. Forest split can be applied to all the components of a bitstream.

Forest Split #1	Forest Split #2	...	Forest Split #nFS

Each connected component data block is composed of three records, the Vertex Graph record, the Triangle Tree record, and the Triangle Data record.

Vertex Graph	Triangle Tree	Triangle Data

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The triangle tree record contains the structure of a triangle spanning tree which links all the triangles of the corresponding connected component forming a simple polygon. The 3D mesh is represented in a triangulated form in the bitstream, which also contains the information necessary to reconstruct the original faces. The vertex graph record contains the information necessary to stitch pairs of boundary edges of the simple polygon to reconstruct the original connectivity, not only within the current connected component, but also to previously decoded connected components. The connectivity information is categorized as *global* information (per connected component) and *local* information (per triangle). The global information is stored in the Vertex Graph and Triangle Tree records. The local information is stored in the Triangle Data record. The triangle data is arranged on a per triangle basis, where the ordering of the triangles is determined by the traversal of the triangle tree.

Data for triangle #1	Data for triangle #2	...	Data for triangle #nT

The data for a given triangle is organized as follows:

marching edge	td_orientation	polygon_edge	coord	normal	color	texCoord

The *marching edge, td_orientation* and *polygon_edge* constitute the per triangle connectivity information. The other fields contain information to reconstruct the vertex coordinates (*coord*) and optionally, normal, color, and texture coordinate (*texCoord*) information.

If the 3D mesh is encoded in hierarchical mode, each Forest Split data block is composed of an optional pre-smoothing data block, an optional post-smoothing data block, a pre-update data block, an optional smoothing constraints data block, a post-update data block, and an optional other-update data block.

Pre-smoothing	Post-smoothing	Pre-update	Constraints	Post-update	Other-update

The pre-smoothing data block contains the parameters used to apply a smoothing step as a global predictor for vertex coordinates. The post-smoothing data block contains the parameters used to apply a smoothing step to the vertex coordinates to remove quantisation artifacts. The pre update data block contains the information necessary to update the connectivity, and the property data updates for properties bound per-face and per-corner to the new faces created by the connectivity update. The smoothing constraints data block contains information used to perform the smoothing steps with sharp edge discontinuities and fixed vertices. After the connectivity update is applied, the pre-smoothing operation specified by the parameters stored in the pre-smoothing data block is applied as a global predictor for the vertex coordinates. The post update data block contains tree loop vertex coordinate updates, with respect to the vertex coordinates predicted by the pre-smoothing step, if applied; normal, color, and texture coordinate updates for properties bound per-vertex to tree loop vertices; normal, and color updates for properties bound per-face to tree loop faces; and normal, color, and texture coordinate updates for properties bound per-corner to tree loop corners. The other-update data block contains vertex coordinate updates and property updates for all the vertices, faces, and corners not updated by data included in the post-update data block.

6.2 Visual bitstream syntax

6.2.1 Start codes and bitstream structure

Start codes are specific bit patterns that do not otherwise occur in the video stream.

Each start code consists of a start code prefix followed by a start code value. The start code prefix is a string of twenty three bits with the value zero followed by a single bit with the value one. The start code prefix is thus the bit string ‘0000 0000 0000 0000 0000 0001’.

The start code value is an eight bit integer which identifies the type of start code. Many types of start code have just one start code value. However video_object_start_code and video_object_layer_start_code are represented by many start code values.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

All start codes shall be byte aligned. This shall be achieved by first inserting a bit with the value zero and then, if necessary, inserting bits with the value one before the start code prefix such that the first bit of the start code prefix is the first (most significant) bit of a byte. For stuffing of 1 to 8 bits, the codewords are as follows in Table 6-2.

Table 6-2-- Stuffing codewords

Bits to be stuffed	Stuffing Codeword
1	0
2	01
3	011
4	0111
5	01111
6	011111
7	0111111
8	01111111

Table 6-3 defines the start code values for all start codes used in the visual bitstream.

Table 6-3 — Start code values

name	**start code value (hexadecimal)**
video_object_start_code	00 through 1F
video_object_layer_start_code	20 through 2F
reserved	30 through 3F
fgs_bp_start_code	40 through 5F
reserved	60 through AF
visual_object_sequence_start_code	B0
visual_object_sequence_end_code	B1
user_data_start_code	B2
group_of_vop_start_code	B3
video_session_error_code	B4
visual_object_start_code	B5
vop_start_code	B6
slice_start_code	B7
extension_start_code	B8
fgs_vop_start_code	B9
fba_object_start_code	BA
fba_object_plane_start_code	BB
mesh_object_start_code	BC
mesh_object_plane_start_code	BD
still_texture_object_start_code	BE
texture_spatial_layer_start_code	BF
texture_snr_layer_start_code	C0
texture_tile_start_code	C1
texture_shape_layer_start_code	C2
stuffing_start_code	C3
reserved	C4-C5
System start codes (see note)	C6 through FF
NOTE System start codes are defined in ISO/IEC 14496-1:2001	

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The use of the start codes is defined in the following syntax description with the exception of the video_session_error_code. The video_session_error_code has been allocated for use by a media interface to indicate where uncorrectable errors have been detected.

This syntax for visual bitstreams defines two types of information:

1. Configuration information

a. Global configuration information, referring to the whole group of visual objects that will be simultaneously decoded and composited by a decoder (VisualObjectSequence()).
b. Object configuration information, referring to a single visual object (VO). This is associated with VisualObject().
c. Object layer configuration information, referring to a single layer of a single visual object (VOL) VideoObjectLayer().

2. Elementary stream data, containing the data for a single layer of a visual object.

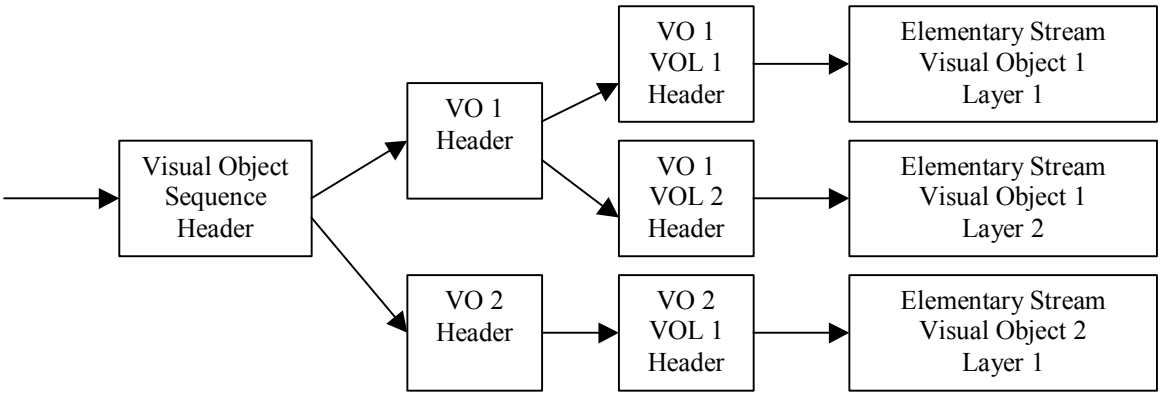


Figure 6-19 — Example Visual Information – Logical Structure

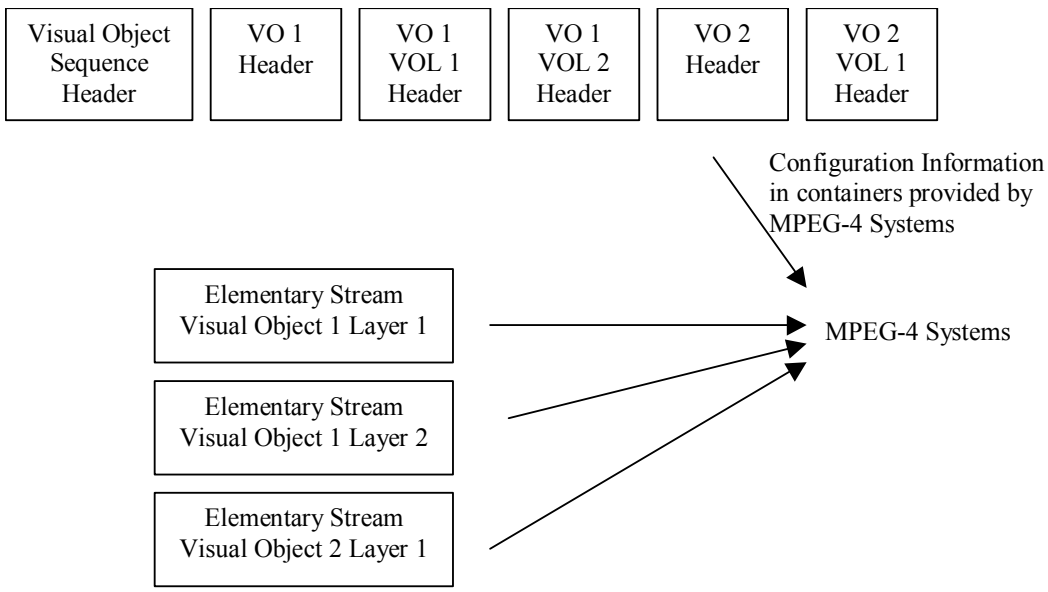


Figure 6-20 — Example Visual Bitstream – Separate Configuration Information / Elementary Stream

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

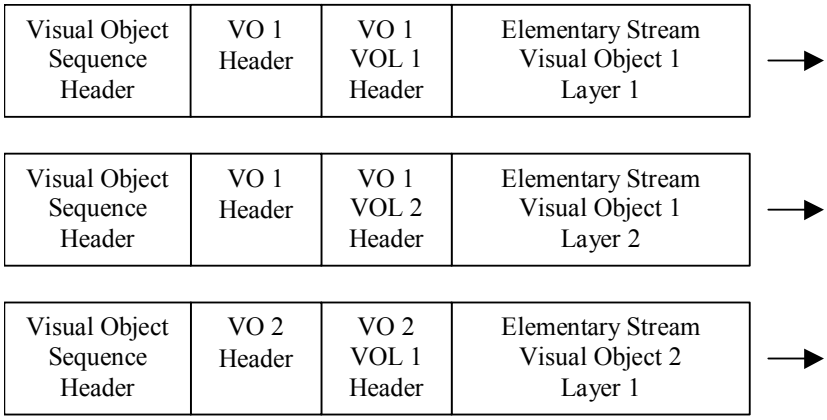


Figure 6-21 — Example Visual Bitstream – Combined Configuration Information / Elementary Stream

The following functions are entry points for elementary streams, and entry into these functions defines the breakpoint between configuration information and elementary streams:

1. Group_of_VideoObjectPlane(),
2. VideoObjectPlane(),
3. video_plane_with_short_header(),
4. MeshObject(),
5. fba_object().

For still texture objects, configuration information ends and elementary stream data begins in StilTextureObject() immediately before the first call to wavelet_dc_decode(), as indicated by the comment in subclause 6.2.8.

There is no overlap of syntax between configuration information and elementary streams.

The configuration information contains all data that is not part of an elementary stream, including that defined by VisualObjectSequence(), VisualObject() and VideoObjectLayer().

This part of ISO/IEC 14496 does not provide for the multiplexing of multiple elementary streams into a single bitstream. One visual bitstream contains exactly one elementary stream, which describes one layer of one visual object. A visual decoder must conceptually have a separate entry port for each layer of each object to be decoded.

Visual objects coded in accordance with this Part may be carried within a Systems bitstream as defined by ISO/IEC 14496-1:2001. The coded visual objects may also be free standing or carried within other types of systems. Configuration information may be carried separately from or combined with elementary stream data:

1. *Separate Configuration / Elementary Streams (e.g. Inside ISO/IEC 14496-1 Bitstreams)*

When coded visual objects are carried within a Systems bitstream defined by ISO/IEC 14496-1:2001, configuration information and elementary stream data are always carried separately. Configuration information and elementary streams follow the syntax below, subject to the break points between them defined above. The Systems specification ISO/IEC 14496-1:2001 defines containers that are used to carry Visual Object Sequence, Visual Object and Video Object Layer configuration information. For video objects one container is used for each layer for each object. This container carries a Visual Object Sequence header, a Visual Object header and a Video Object Layer header. For other types of visual objects, one container per visual object is used. This container carries a Visual Object Sequence header and a Visual Object header. The Visual Object Sequence Header must be identical for all visual streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical.

2. *Combined Configuration / Elementary Streams*

The elementary stream data associated with a single layer may be wrapped in configuration information defined in accordance with the syntax below. A visual bitstream may contain at most one instance of each of

--`,,`,,`,,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

VisualObjectSequence(), VisualObject() and VideoObjectLayer(), with the exception of repetition of the Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header as described below. The Visual Object Sequence Header must be identical for all visual streams input simultaneously to a decoder. The Visual Object Headers for each layer of a multilayer object must be identical.

The Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header may be repeated in a single visual bitstream. Repeating these headers enables random access into the visual bitstream and recovery of these headers when the original headers are corrupted by errors. This header repetition is used only when visual_object_type in the Visual Object Header indicates that visual object type is video. (i.e. visual_object_type=="video ID") All of the data elements in the Visual Object Sequence Header, the Visual Object Header and the Video Object Layer Header repeated in a visual bitstream shall have the same value as in the original headers, except that first_half_vbv_occupancy and latter_half_vbv_occupancy may be changed to specify the VBV occupancy just before the removal of the first VOP following the repeated Video Object Layer Header.

6.2.2 Visual Object Sequence and Visual Object

VisualObjectSequence() {	**No. of bits**	**Mnemonic**
do {		
visual_object_sequence_start_code	32	bslbf
profile_and_level_indication	8	uimsbf
if (profile_and_level_indication == 11100001-11101000) {		
next_start_code_studio()		
extension_and_user_data(0)		
StudioVisualObject()		
} else {		
while (next_bits()== user_data_start_code){		
user_data()		
}		
VisualObject()		
}		
} while (next_bits() != visual_object_sequence_end_code)		
visual_object_sequence_end_code	32	bslbf
}		

VisualObject() {	**No. of bits**	**Mnemonic**
visual_object_start_code	32	bslbf
is_visual_object_identifier	1	uimsbf
if (is_visual_object_identifier) {		
visual_object_verid	4	uimsbf
visual_object_priority	3	uimsbf
}		
visual_object_type	4	uimsbf
if (visual_object_type == "video ID" || visual_object_type == "still texture ID") {		
video_signal_type()		
}		
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (visual_object_type == “video ID”) {		
video_object_start_code	32	bslbf
VideoObjectLayer()		
}		
else if (visual_object_type == “still texture ID”) {		
StillTextureObject()		
}		
else if (visual_object_type == “mesh ID”) {		
MeshObject()		
}		
else if (visual_object_type == “FBA ID”) {		
FBAObject()		
}		
else if (visual_object_type == “3D mesh ID”) {		
3D_Mesh_Object()		
}		
if (next_bits() != “0000 0000 0000 0000 0000 0001”)		
next_start_code()		
}		

video_signal_type() {	**No. of bits**	**Mnemonic**
video_signal_type	1	bslbf
if (video_signal_type) {		
video_format	3	uimsbf
video_range	1	bslbf
colour_description	1	bslbf
if (colour_description) {		
colour_primaries	8	uimsbf
transfer_characteristics	8	uimsbf
matrix_coefficients	8	uimsbf
}		
}		
}		

6.2.2.1 User data

user_data() {	**No. of bits**	**Mnemonic**
user_data_start_code	32	bslbf
while(next_bits() != ‘0000 0000 0000 0000 0000 0001’) {		
user_data	8	uimsbf
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.3 Video Object Layer

VideoObjectLayer() {	No. of bits	Mnemonic
if(next_bits() == video_object_layer_start_code) {		
short_video_header = 0		
video_object_layer_start_code	32	bslbf
random_accessible_vol	1	bslbf
video_object_type_indication	8	uimsbf
if (video_object_type_indication == “Fine Granularity Scalable”) {		
fgs_layer_type	2	uimsbf
video_object_layer_priority	3	uimsbf
aspect_ratio_info	4	uimsbf
if (aspect_ratio_info == “extended_PAR”) {		
par_width	8	uimsbf
par_height	8	uimsbf
}		
vol_control_parameters	1	bslbf
if (vol_control_parameters) {		
chroma_format	2	uimsbf
low_delay	1	uimsbf
}		
marker_bit	1	bslbf
vop_time_increment_resolution	16	uimsbf
marker_bit	1	bslbf
fixed_vop_rate	1	bslbf
if (fixed_vop_rate)		
fixed_vop_time_increment	1-16	uimsbf
marker_bit	1	bslbf
video_object_layer_width	13	uimsbf
marker_bit	1	bslbf
video_object_layer_height	13	uimsbf
marker_bit	1	bslbf
interlaced	1	bslbf
if (fgs_layer_type ==“FGST” || fgs_layer_type ==“FGS_FGST”)		
fgs_ref_layer_id	4	uimsbf
if (fgs_layer_type ==“FGS” || fgs_layer_type ==“FGS_FGST”) {		
fgs_frequency_weighting_enable	1	bslbf
if (fgs_frequency_weighting_enable) {		
load_fgs_frequency_weighting_matrix	1	bslbf
if (load_fgs_frequency_weighting_matrix)		
fgs_frequency_weighting_matrix	3*[2-64]	uimsbf
}		
}		
if (fgs_layer_type ==“FGST” || fgs_layer_type ==“FGS_FGST”) {		
fgst_frequency_weighting_enable	1	bslbf

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

if (fgst_frequency_weighting_enable) {		
load_fgst_frequency_weighting_matrix	1	bslbf
if (load_fgst_frequency_weighting_matrix)		
fgst_frequency_weighting_matrix	3*[2-64]	uimsbf
}		
}		
quarter_sample	1	bslbf
fgs_resync_marker_disable	1	bslbf
do {		
if (nextbits_bytealigned() == group_of_vop_start_code)		
Group_of_VideoObjectPlane()		
FGSVideoObjectPlane()		
} while((nextbits_bytealigned()==group_of_vop_start_code)|| (nextbits_bytealigned()==fgs_vop_start_code))		
} else {		
is_object_layer_identifier	1	uimsbf
if (is_object_layer_identifier) {		
video_object_layer_verid	4	uimsbf
video_object_layer_priority	3	uimsbf
}		
aspect_ratio_info	4	uimsbf
if (aspect_ratio_info == “extended_PAR”) {		
par_width	8	uimsbf
par_height	8	uimsbf
}		
vol_control_parameters	1	bslbf
if (vol_control_parameters) {		
chroma_format	2	uimsbf
low_delay	1	uimsbf
vbv_parameters	1	blsbf
if (vbv_parameters) {		
first_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
latter_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
first_half_vbv_buffer_size	15	uimsbf
marker_bit	1	bslbf
latter_half_vbv_buffer_size	3	uimsbf
first_half_vbv_occupancy	11	uimsbf
marker_bit	1	blsbf
latter_half_vbv_occupancy	15	uimsbf
marker_bit	1	blsbf
}		
}		
video_object_layer_shape	2	uimsbf
if (video_object_layer_shape == "grayscale" && video_object_layer_verid != ‘0001’)		
video_object_layer_shape_extension	4	uimsbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

marker_bit	1	bslbf
vop_time_increment_resolution	16	uimsbf
marker_bit	1	bslbf
fixed_vop_rate	1	bslbf
if (fixed_vop_rate)		
fixed_vop_time_increment	1-16	uimsbf
if (video_object_layer_shape != “binary only”) {		
if (video_object_layer_shape == “rectangular”) {		
marker_bit	1	bslbf
video_object_layer_width	13	uimsbf
marker_bit	1	bslbf
video_object_layer_height	13	uimsbf
marker_bit	1	bslbf
}		
interlaced	1	bslbf
obmc_disable	1	bslbf
if (video_object_layer_verid == ‘0001’)		
sprite_enable	1	bslbf
else		
sprite_enable	2	uimsbf
if (sprite_enable== “static” || sprite_enable == “GMC”) {		
if (sprite_enable != “GMC”) {		
sprite_width	13	uimsbf
marker_bit	1	bslbf
sprite_height	13	uimsbf
marker_bit	1	bslbf
sprite_left_coordinate	13	simsbf
marker_bit	1	bslbf
sprite_top_coordinate	13	simsbf
marker_bit	1	bslbf
}		
no_of_sprite_warping_points	6	uimsbf
sprite_warping_accuracy	2	uimsbf
sprite_brightness_change	1	bslbf
if (sprite_enable != “GMC”)		
low_latency_sprite_enable	1	bslbf
}		
if (video_object_layer_verid != ‘0001’ && video_object_layer_shape != ”rectangular”)		
sadct_disable	1	bslbf
not_8_bit	1	bslbf
if (not_8_ bit) {		
quant_precision	4	uimsbf
bits_per_pixel	4	uimsbf
}		
if (video_object_layer_shape==”grayscale”) {		
no_gray_quant_update	1	bslbf
composition_method	1	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

linear_composition	1	bslbf
}		
quant_type	1	bslbf
if (quant_type) {		
load_intra_quant_mat	1	bslbf
if (load_intra_quant_mat)		
intra_quant_mat	8*[2-64]	uimsbf
load_nonintra_quant_mat	1	bslbf
if (load_nonintra_quant_mat)		
nonintra_quant_mat	8*[2-64]	uimsbf
if(video_object_layer_shape==”grayscale”) {		
for(i=0; i<aux_comp_count; i++) {		
load_intra_quant_mat_grayscale	1	bslbf
if(load_intra_quant_mat_grayscale)		
intra_quant_mat_grayscale[i]	8*[2-64]	uimsbf
load_nonintra_quant_mat_grayscale	1	bslbf
if(load_nonintra_quant_mat_grayscale)		
nonintra_quant_mat_grayscale[i]	8*[2-64]	uimsbf
}		
}		
}		
if (video_object_layer_verid != ‘0001’)		
quarter_sample	1	bslbf
complexity_estimation_disable	1	bslbf
if (!complexity_estimation_disable)		
define_vop_complexity_estimation_header()		
resync_marker_disable	1	bslbf
data_partitioned	1	bslbf
if(data_partitioned)		
reversible_vlc	1	bslbf
if(video_object_layer_verid != ’0001’) {		
newpred_enable	1	bslbf
if (newpred_enable) {		
requested_upstream_message_type	2	uimsbf
newpred_segment_type	1	bslbf
}		
reduced_resolution_vop_enable	1	bslbf
}		
scalability	1	bslbf
if (scalability) {		
hierarchy_type	1	bslbf
ref_layer_id	4	uimsbf
ref_layer_sampling_direc	1	bslbf
hor_sampling_factor_n	5	uimsbf
hor_sampling_factor_m	5	uimsbf
vert_sampling_factor_n	5	uimsbf
vert_sampling_factor_m	5	uimsbf
enhancement_type	1	bslbf

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if(video_object_layer_shape == “binary” && hierarchy_type== ‘0’) {		
use_ref_shape	1	bslbf
use_ref_texture	1	bslbf
shape_hor_sampling_factor_n	5	uimsbf
shape_hor_sampling_factor_m	5	uimsbf
shape_vert_sampling_factor_n	5	uimsbf
shape_vert_sampling_factor_m	5	uimsbf
}		
}		
}		
else {		
if(video_object_layer_verid !=”0001”) {		
scalability	1	bslbf
if(scalability) {		
ref_layer_id	4	uimsbf
shape_hor_sampling_factor_n	5	uimsbf
shape_hor_sampling_factor_m	5	uimsbf
shape_vert_sampling_factor_n	5	uimsbf
shape_vert_sampling_factor_m	5	uimsbf
}		
}		
resync_marker_disable	1	bslbf
}		
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		
if (sprite_enable == “static” && !low_latency_sprite_enable)		
VideoObjectPlane()		
do {		
if (next_bits() == group_of_vop_start_code)		
Group_of_VideoObjectPlane()		
VideoObjectPlane()		
if ((preceding_vop_coding_type == "B" || preceding_vop_coding_type == "S" || video_object_layer_shape != "rectangular") && next_bits() == stuffing_start_code) {		
stuffing_start_code	32	bslbf
while (next_bits() != ‘0000 0000 0000 0000 0000 0001’)		
stuffing_byte	8	bslbf
}		
} while ((next_bits() == group_of_vop_start_code) || (next_bits() == vop_start_code))		
}		
} else {		
short_video_header = 1		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

do {		
video_plane_with_short_header()		
} while(next_bits() == short_video_start_marker)		
}		
}		
NOTE — preceding_vop_coding_type has the same value as vop_coding_type in the immediately preceding VideoObjectPlane() in the decoding order.		

define_vop_complexity_estimation_header() {	**No. of bits**	**Mnemonic**
estimation_method	2	uimsbf
if (estimation_method ==’00’ || estimation_method == ‘01’) {		
shape_complexity_estimation_disable	1	
if (!shape_complexity_estimation_disable) {		bslbf
opaque	1	bslbf
transparent	1	bslbf
intra_cae	1	bslbf
inter_cae	1	bslbf
no_update	1	bslbf
upsampling	1	bslbf
}		
texture_complexity_estimation_set_1_disable	1	bslbf
if (!texture_complexity_estimation_set_1_disable) {		
intra_blocks	1	bslbf
inter_blocks	1	bslbf
inter4v_blocks	1	bslbf
not_coded_blocks	1	bslbf
}		
marker_bit	1	bslbf
texture_complexity_estimation_set_2_disable	1	bslbf
if (!texture_complexity_estimation_set_2_disable) {		
dct_coefs	1	bslbf
dct_lines	1	bslbf
vlc_symbols	1	bslbf
vlc_bits	1	bslbf
}		
motion_compensation_complexity_disable	1	bslbf
iIf (!motion_compensation_complexity_disable) {		
apm	1	bslbf
npm	1	bslbf
interpolate_mc_q	1	bslbf
forw_back_mc_q	1	bslbf
halfpel2	1	bslbf
halfpel4	1	bslbf
}		
marker_bit	1	bslbf
if(estimation_method == ‘01’) {		
version2_complexity_estimation_disable	1	bslbf

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (!version2_complexity_estimation_disable) {		
sadct	1	bslbf
quarterpel	1	bslbf
}		
}		
}		
}		

6.2.4 Group of Video Object Plane

Group_of_VideoObjectPlane() {	**No. of bits**	**Mnemonic**
group_of_vop_start_code	32	bslbf
time_code	18	
closed_gov	1	bslbf
broken_link	1	bslbf
next_start_code()		
while (next_bits()== user_data_start_code){		
user_data()		
}		
}		

6.2.5 Video Object Plane and Video Plane with Short Header

VideoObjectPlane() {	**No. of bits**	**Mnemonic**
vop_start_code	32	bslbf
vop_coding_type	2	uimsbf
do {		
modulo_time_base	1	bslbf
} while (modulo_time_base != '0')		
marker_bit	1	bslbf
vop_time_increment	1-16	uimsbf
marker_bit	1	bslbf
vop_coded	1	bslbf
if (vop_coded == '0') {		
next_start_code()		
return()		
}		
if (newpred_enable) {		
vop_id	4-15	uimsbf
vop_id_for_prediction_indication	1	bslbf
if (vop_id_for_prediction_indication)		
vop_id_for_prediction	4-15	uimsbf
marker_bit	1	bslbf
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if ((video_object_layer_shape != “binary only”) && (vop_coding_type == “P” || (vop_coding_type == “S” && sprite_enable == “GMC”)))		
vop_rounding_type	1	bslbf
if ((reduced_resolution_vop_enable) && (video_object_layer_shape == “rectangular”) && ((vop_coding_type == “P”) || (vop_coding_type == “I”)))		
vop_reduced_resolution	1	bslbf
if (video_object_layer_shape != “rectangular”) {		
if(!(sprite_enable == “static” && vop_coding_type == “I”)) {		
vop_width	13	uimsbf
marker_bit	1	bslbf
vop_height	13	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
}		
if ((video_object_layer_shape != “ binary only”) && scalability && enhancement_type)		
background_composition	1	bslbf
change_conv_ratio_disable	1	bslbf
vop_constant_alpha	1	bslbf
if (vop_constant_alpha)		
vop_constant_alpha_value	8	bslbf
}		
if (video_object_layer_shape != ‘’binary only‘’)		
if (!complexity_estimation_disable)		
read_vop_complexity_estimation_header()		
if (video_object_layer_shape != “binary only”) {		
intra_dc_vlc_thr	3	uimsbf
if (interlaced) {		
top_field_first	1	bslbf
alternate_vertical_scan_flag	1	bslbf
}		
}		
if ((sprite_enable ==“static” || sprite_enable==“GMC”) && vop_coding_type == “S”) {		
if (no_of_sprite_warping_points > 0)		
sprite_trajectory()		
if (sprite_brightness_change)		
brightness_change_factor()		
if(sprite_enable == “static”) {		
if (sprite_transmit_mode != “stop” && low_latency_sprite_enable) {		
do {		
sprite_transmit_mode	2	uimsbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if ((sprite_transmit_mode == “piece”) || (sprite_transmit_mode == “update”))		
decode_sprite_piece()		
} while (sprite_transmit_mode != “stop” && sprite_transmit_mode != “pause”)		
}		
next_start_code()		
return()		
}		
}		
if (video_object_layer_shape != “binary only”) {		
vop_quant	3-9	uimsbf
if(video_object_layer_shape==”grayscale”)		
for(i=0; i<aux_comp_count; i++)		
vop_alpha_quant[i]	6	uimsbf
if (vop_coding_type != “I”)		
vop_fcode_forward	3	uimsbf
if (vop_coding_type == “B”)		
vop_fcode_backward	3	uimsbf
if (!scalability) {		
if (video_object_layer_shape != “rectangular” && vop_coding_type != “I”)		
vop_shape_coding_type	1	bslbf
motion_shape_texture()		
while (nextbits_bytealigned() == resync_marker) {		
video_packet_header()		
motion_shape_texture()		
}		
}		
else {		
if (enhancement_type) {		
load_backward_shape	1	bslbf
if (load_backward_shape) {		
backward_shape_width	13	uimsbf
marker_bit	1	bslbf
backward_shape_ height	13	uimsbf
marker_bit	1	bslbf
backward_shape_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
backward_shape_vertical_mc_spatial_ref	13	simsbf
backward_shape()		
load_forward_shape	1	bslbf
if (load_forward_shape) {		
forward_shape_width	13	uimsbf
marker_bit	1	bslbf
forward_shape_height	13	uimsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

forward_shape_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
forward_shape_vertical_mc_spatial_ref	13	simsbf
forward_shape()		
}		
}		
}		
ref_select_code	2	uimsbf
combined_motion_shape_texture()		
}		
}		
else {		
combined_motion_shape_texture()		
while (nextbits_bytealigned() == resync_marker) {		
video_packet_header()		
combined_motion_shape_texture()		
}		
}		
next_start_code()		
}		

6.2.5.1 Complexity Estimation Header

read_vop_complexity_estimation_header() {		**No. of bits**	**Mnemonic**
if (estimation_method==’00’ || estimation_method==’01’) {			
if (vop_coding_type==“I”) {			
if (opaque)	**dcecs_opaque**	8	uimsbf
if (transparent)	**dcecs_transparent**	8	uimsbf
if (intra_cae)	**dcecs_intra_cae**	8	uimsbf
if (inter_cae)	**dcecs_inter_cae**	8	uimsbf
if (no_update)	**dcecs_no_update**	8	uimsbf
if (upsampling)	**dcecs_upsampling**	8	uimsbf
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded_blocks)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf
if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (sadct)	**dcecs_sadct**	8	uimsbf
}			
if (vop_coding_type==“P”) {			
if (opaque)	**dcecs_opaque**	8	uimsbf
if (transparent)	**dcecs_transparent**	8	uimsbf
if (intra_cae)	**dcecs_intra_cae**	8	uimsbf
if (inter_cae)	**dcecs_inter_cae**	8	uimsbf

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (no_update)	**dcecs_no_update**	8	uimsbf
if (upsampling)	**dcecs_upsampling**	8	uimsbf
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf
if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (inter_blocks)	**dcecs_inter_blocks**	8	uimsbf
if (inter4v_blocks)	**dcecs_inter4v_blocks**	8	uimsbf
if (apm)	**dcecs_apm**	8	uimsbf
if (npm)	**dcecs_npm**	8	uimsbf
if (forw_back_mc_q)	**dcecs_forw_back_mc_q**	8	uimsbf
if (halfpel2)	**dcecs_halfpel2**	8	uimsbf
if (halfpel4)	**dcecs_halfpel4**	8	uimsbf
if (sadct)	**dcecs_sadct**	8	uimsbf
if (quarterpel)	**dcecs_quarterpel**	8	uimsbf
}			
if (vop_coding_type==“B”) {			
if (opaque)	**dcecs_opaque**	8	uimsbf
if (transparent)	**dcecs_transparent**	8	uimsbf
if (intra_cae)	**dcecs_intra_cae**	8	uimsbf
if (inter_cae)	**dcecs_inter_cae**	8	uimsbf
if (no_update)	**dcecs_no_update**	8	uimsbf
if (upsampling)	**dcecs_upsampling**	8	uimsbf
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded_blocks)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf
if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (inter_blocks)	**dcecs_inter_blocks**	8	uimsbf
if (inter4v_blocks)	**dcecs_inter4v_blocks**	8	uimsbf
if (apm)	**dcecs_apm**	8	uimsbf
if (npm)	**dcecs_npm**	8	uimsbf
if (forw_back_mc_q)	**dcecs_forw_back_mc_q**	8	uimsbf
if (halfpel2)	**dcecs_halfpel2**	8	uimsbf
if (halfpel4)	**dcecs_halfpel4**	8	uimsbf
if (interpolate_mc_q)	**dcecs_interpolate_mc_q**	8	uimsbf
if (sadct)	**dcecs_sadct**	8	uimsbf
if (quarterpel)	**dcecs_quarterpel**	8	uimsbf
}			
if (vop_coding_type==‘S’&& sprite_enable == “static”) {			
if (intra_blocks)	**dcecs_intra_blocks**	8	uimsbf
if (not_coded_blocks)	**dcecs_not_coded_blocks**	8	uimsbf
if (dct_coefs)	**dcecs_dct_coefs**	8	uimsbf
if (dct_lines)	**dcecs_dct_lines**	8	uimsbf

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

if (vlc_symbols)	**dcecs_vlc_symbols**	8	uimsbf
if (vlc_bits)	**dcecs_vlc_bits**	4	uimsbf
if (inter_blocks)	**dcecs_inter_blocks**	8	uimsbf
if (inter4v_blocks)	**dcecs_inter4v_blocks**	8	uimsbf
if (apm)	**dcecs_apm**	8	uimsbf
if (npm)	**dcecs_npm**	8	uimsbf
if (forw_back_mc_q)	**dcecs_forw_back_mc_q**	8	uimsbf
if (halfpel2)	**dcecs_halfpel2**	8	uimsbf
if (halfpel4)	**dcecs_halfpel4**	8	uimsbf
if (interpolate_mc_q)	**dcecs_interpolate_mc_q**	8	uimsbf
}			
}			
}			

6.2.5.2 Video Plane with Short Header

video_plane_with_short_header() {	**No. of bits**	**Mnemonic**
short_video_start_marker	22	bslbf
temporal_reference	8	uimsbf
marker_bit	1	bslbf
zero_bit	1	bslbf
split_screen_indicator	1	bslbf
document_camera_indicator	1	bslbf
full_picture_freeze_release	1	bslbf
source_format	3	bslbf
picture_coding_type	1	bslbf
four_reserved_zero_bits	4	bslbf
vop_quant	5	uimsbf
zero_bit	1	bslbf
do{		
pei	1	bslbf
if (pei == “1”)		
psupp	8	bslbf
} while (pei == “1”)		
gob_number = 0		
for(i=0; i<num_gobs_in_vop; i++)		
gob_layer()		
if(next_bits() == short_video_end_marker)		
short_video _end_marker	22	uimsbf
while(!bytealigned())		
zero_bit	1	bslbf
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

gob_layer() {	No. of bits	Mnemonic
gob_header_empty = 1		
if(gob_number != 0) {		
if (next_bits() == gob_resync_marker) {		
gob_header_empty = 0		
gob_resync_marker	17	bslbf
gob_number	5	uimsbf
gob_frame_id	2	bslbf
quant_scale	5	uimsbf
}		
}		
for(i=0; i<num_macroblocks_in_gob; i++)		
macroblock()		
if(next_bits() != gob_resync_marker && nextbits_bytealigned() == gob_resync_marker)		
while(!bytealigned())		
zero_bit	1	bslbf
gob_number++		
}		

video_packet_header() {	No. of bits	Mnemonic
next_resync_marker()		
resync_marker	17-23	uimsbf
if (video_object_layer_shape != “rectangular”) {		
header_extension_code	1	bslbf
if (header_extension_code && !(sprite_enable = “static“ && vop_coding_type == “I”)) {		
vop_width	13	uimsbf
marker_bit	1	bslbf
vop_height	13	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
vop_vertical_mc_spatial_ref	13	simsbf
marker_bit	1	bslbf
}		
}		
macroblock_number	1-14	vlclbf
if (video_object_layer_shape != “binary only”)		
quant_scale	3-9	uimsbf
if (video_object_layer_shape == “rectangular”)		
header_extension_code	1	bslbf
if (header_extension_code) {		
do {		
modulo_time_base	1	bslbf
} while (modulo_time_base != ‘0’)		
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

vop_time_increment	1-16	bslbf
marker_bit	1	bslbf
vop_coding_type	2	uimsbf
if (video_object_layer_shape != “rectangular”) {		
change_conv_ratio_disable	1	bslbf
if (vop_coding_type != “I”)		
vop_shape_coding_type	1	bslbf
}		
if (video_object_layer_shape != “binary only”) {		
intra_dc_vlc_thr	3	uimsbf
if (sprite_enable == “GMC” && vop_coding_type == “S” && no_of_sprite_warping_points > 0)		
sprite_trajectory()		
if ((reduced_resolution_vop_enable) && (video_object_layer_shape == “rectangular”) && ((vop_coding_type == “P”) || (vop_coding_type == “I”)))		
vop_reduced_resolution	1	bslbf
if (vop_coding_type != “I”)		
vop_fcode_forward	3	uimsbf
if (vop_coding_type == “B”)		
vop_fcode_backward	3	uimsbf
}		
}		
if (newpred_enable) {		
vop_id	4-15	uimsbf
vop_id_for_prediction_indication	1	bslbf
if (vop_id_for_prediction_indication)		
vop_id_for_prediction	4-15	uimsbf
marker_bit	1	bslbf
}		
}		

6.2.5.3 Motion Shape Texture

motion_shape_texture() {	**No. of bits**	**Mnemonic**
if (data_partitioned)		
data_partitioned_motion _shape_texture()		
else		
combined_motion_shape_texture()		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

combined_motion_shape_texture() {	No. of bits	Mnemonic
do{		
macroblock()		
} while ((nextbits_bytealigned() != resync_marker && nextbits_bytealigned() != '000 0000 0000 0000 0000 0000') || valid_stuffing_bits() == 0)		
}		

data_partitioned_motion_shape_texture() {	No. of bits	Mnemonic
if (vop_coding_type == "I") {		
data_partitioned_i_vop()		
} else if (vop_coding_type == "P" || (vop_coding_type == "S" && sprite_enable == "GMC")) {		
data_partitioned_p_vop()		
} else if (vop_coding_type == "B") {		
combined_motion_shape_texture()		
}		
NOTE: Data partitioning is not supported in B-VOPs.		

data_partitioned_i_vop() {	No. of bits	Mnemonic
do{		
if (video_object_layer_shape != "rectangular"){		
bab_type	1-3	vlclbf
if (bab_type >= 4) {		
if (!change_conv_rate_disable)		
conv_ratio	1-2	vlclbf
scan_type	1	bslbf
binary_arithmetic_code()		
}		
}		
if (!transparent_mb()) {		
if (video_object_layer_shape != "rectangular") {		
do {		
mcbpc	1-9	vlclbf
} while (derived_mb_type == "stuffing")		
} else {		
mcbpc	1-9	vlclbf
if (derived_mb_type == "stuffing")		
continue		
}		
if (mb_type == 4)		
dquant	2	bslbf
if (use_intra_dc_vlc) {		
for (j = 0; j < 4; j++) {		
if (!transparent_block(j)) {		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

dct_dc_size_luminance	2-11	vlclbf
if (dct_dc_size_luminance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
}		
}		
for (j = 0; j < 2; j++) {		
dct_dc_size_chrominance	2-12	vlclbf
if (dct_dc_size_chrominance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
}		
}		
} while (next_bits() != dc_marker)		
dc_marker /* 110 1011 0000 0000 0001 */	19	bslbf
for (i = 0; i < mb_in_video_packet; i++) {		
if (!transparent_mb()) {		
ac_pred_flag	1	bslbf
cbpy	1-6	vlclbf
}		
}		
for (i = 0; i < mb_in_video_packet; i++) {		
if (!transparent_mb()) {		
for (j = 0; j < block_count; j++)		
block(j)		
}		
}		
}		

NOTE 1: The value of mb_in_video_packet is the number of macroblocks in a video packet. The count of stuffing macroblocks is not included in this value.

NOTE 2: The value of block_count is 6 in the 4:2:0 format.

NOTE 3: The value of alpha_block_count is 4.

data_partitioned_p_vop() {	**No. of bits**	**Mnemonic**
do{		
if (video_object_layer_shape != “rectangular”){		
bab_type	1-7	vlclbf
if ((bab_type == 1) || (bab_type == 6)) {		
mvds_x	1-18	vlclbf
mvds_y	1-18	vlclbf
}		
if (bab_type >= 4) {		
if (!change_conv_rate_disable)		
conv_ratio	1-2	vlclbf

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

scan_type	1	bslbf
binary_arithmetic_code()		
}		
}		
if (!transparent_mb()) {		
if (video_object_layer_shape != “rectangular”) {		
do {		
not_coded	1	bslbf
if (!not_coded)		
mcbpc	1-9	vlclbf
} while (!(not_coded || derived_mb_type != “stuffing”))		
} else {		
not_coded	1	bslbf
if (!not_coded) {		
mcbpc	1-9	vlclbf
if (derived_mb_type == “stuffing”)		
continue		
}		
}		
if (!not_coded) {		
if (sprite_enable == “GMC” && vop_coding_type == “S” && derived_mb_type < 2)		
mcsel	1	bslbf
if ((!(sprite_enable == “GMC” && vop_coding_type == “S” && mcsel) && derived_mb_type < 2) || derived_mb_type == 2)		
motion_coding(“forward”, derived_mb_type)		
}		
}		
} while (next_bits() != motion_marker)		
motion_marker /* 1 1111 0000 0000 0001 */	17	bslbf
for (i = 0; i < mb_in_video_packet; i++) {		
if (!transparent_mb()) {		
if (!not_coded){		
if (derived_mb_type >= 3)		
ac_pred_flag	1	bslbf
cbpy	1-6	vlclbf
if (derived_mb_type == 1 || derived_mb_type == 4)		
dquant	2	bslbf
if (derived_mb_type >= 3 && use_intra_dc_vlc) {		
for (j = 0; j < 4; j++) {		
if (!transparent_block(j)) {		
dct_dc_size_luminance	2-11	vlclbf
if (dct_dc_size_luminance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

}		
for (j = 0; j < 2; j++) {		
dct_dc_size_chrominance	2-12	vlclbf
if (dct_dc_size_chrominance > 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
}		
}		
}		
}		
for (i = 0; i < mb_in_video_packet; i++) {		
if (!transparent_mb()) {		
if (! not_coded) {		
for (j = 0; j < block_count; j++)		
block(j)		
}		
}		
}		
}		

NOTE 1: The value of mb_in_video_packet is the number of macroblocks in a video packet. The count of stuffing macroblocks is not included in this value.

NOTE 2: The value of block_count is 6 in the 4:2:0 format.

NOTE 3: The value of alpha_block_count is 4.

motion_coding(mode, type_of_mb) {	**No. of bits**	**Mnemonic**
motion_vector(mode)		
if (type_of_mb == 2) {		
for (i = 0; i < 3; i++)		
motion_vector(mode)		
}		
}		

6.2.5.4 Sprite coding

decode_sprite_piece() {	**No. of bits**	**Mnemonic**
piece_quant	5	bslbf
piece_width	9	bslbf
piece_height	9	bslbf
marker_bit	1	bslbf
piece_xoffset	9	bslbf
piece_yoffset	9	bslbf
sprite_shape_texture()		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sprite_shape_texture() {	No. of bits	Mnemonic
if (sprite_transmit_mode == “piece”) {		
for (i=0; i < piece_height; i++) {		
for (j=0; j < piece_width; j++) {		
if (!send_mb()) {		
macroblock()		
}		
}		
}		
}		
if (sprite_transmit_mode == “update”) {		
for (i=0; i < piece_height; i++) {		
for (j=0; j < piece_width; j++) {		
macroblock()		
}		
}		
}		
}		

sprite_trajectory() {	No. of bits	Mnemonic
for (i=0; i < no_of_sprite_warping_points; i++) {		
warping_mv_code(du[i])		
warping_mv_code(dv[i])		
}		
}		

warping_mv_code(d) {	No. of bits	Mnemonic
dmv_length	2-12	uimsbf
if (dmv_length != ‘00’)		
dmv_code	1-14	uimsbf
marker_bit	1	bslbf
}		

brightness_change_factor() {	No. of bits	Mnemonic
brightness_change_factor_size	1-4	uimsbf
brightness_change_factor_code	5-10	uimsbf
}		

6.2.6 Macroblock

macroblock() {	No. of bits	Mnemonic
if (vop_coding_type != “B”) {		
if (video_object_layer_shape != “rectangular” && !(sprite_enable == “static” && low_latency_sprite_enable && sprite_transmit_mode == “update”))		
mb_binary_shape_coding()		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (video_object_layer_shape != “binary only”) {		
if (!transparent_mb()) {		
if (video_object_layer_shape != “rectangular” && !(sprite_enable = "static" && low_latency_sprite_enable && sprite_transmit_mode == “update”)) {		
do{		
if (vop_coding_type != “I” && !(sprite_enable == "static" && sprite_transmit_mode == “piece”))		
not_coded	1	bslbf
if (!not_coded || vop_coding_type == “I” || (vop_coding_type == "S" && low_latency_sprite_enable && sprite_transmit_mode == "piece"))		
mcbpc	1-9	vlclbf
} while(!(not_coded || derived_mb_type != “stuffing”))		
} else {		
if (vop_coding_type != “I” && !(sprite_enable = "static" && sprite_transmit_mode == “piece”))		
not_coded	1	bslbf
if (!not_coded || vop_coding_type == “I” || (vop_coding_type == "S" && low_latency_sprite_enable && sprite_transmit_mode == "piece"))		
mcbpc	1-9	vlclbf
}		
if (!not_coded || vop_coding_type == “I” || (vop_coding_type == "S" && low_latency_sprite_enable && sprite_transmit_mode == "piece")) {		
if (vop_coding_type == “S” && sprite_enable == “GMC” && (derived_mb_type == 0 || derived_mb_type == 1))		
mcsel	1	bslbf
if (!short_video_header && (derived_mb_type == 3 || derived_mb_type == 4))		
ac_pred_flag	1	bslbf
if (derived_mb_type != “stuffing”)		
cbpy	1-6	vlclbf
else		
return()		
if (derived_mb_type == 1 || derived_mb_type == 4)		
dquant	2	bslbf
if (interlaced)		
interlaced_information()		
if (!(ref_select_code==‘11’ && scalability) && sprite_enable != “static”) {		
if ((derived_mb_type == 0 || derived_mb_type == 1) && (vop_coding_type == “P” || (vop_coding_type == “S” && !mcsel))) {		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

motion_vector(“forward”)		
if (interlaced && field_prediction)		
motion_vector(“forward”)		
}		
if (derived_mb_type == 2) {		
for (j=0; j < 4; j++)		
if (!transparent_block(j))		
motion_vector(“forward”)		
}		
}		
for (i = 0; i < block_count; i++)		
if(!transparent_block(i))		
block(i)		
}		
}		
}		
}		
else {		
if (video_object_layer_shape != “rectangular”)		
mb_binary_shape_coding()		
if ((co_located_not_coded != 1 || (scalability && (ref_select_code != '11' || enhancement_type == 1)) || (sprite_enable == “GMC” && backward_reference_vop_coding_type == “S”)) && video_object_layer_shape != “binary only”) {		
if (!transparent_mb()) {		
modb	1-2	vlclbf
if (modb != ‘1’) {		
mb_type	1-4	vlclbf
if (modb == ‘00’)		
cbpb	3-6	vlclbf
if (ref_select_code != ‘00’ || !scalability) {		
if (mb_type != “1” && cbpb!=0)		
dbquant	1-2	vlclbf
if (interlaced)		
interlaced_information()		
if (mb_type == ‘01’ || mb_type == ‘0001’) {		
motion_vector(“forward”)		
if (interlaced && field_prediction)		
motion_vector(“forward”)		
}		
if (mb_type == ‘01’ || mb_type == ‘001’) {		
motion_vector(“backward”)		
if (interlaced && field_prediction)		
motion_vector(“backward”)		
}		
if (mb_type == “1”)		
motion_vector(“direct”)		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

}		
if (ref_select_code == '00' && scalability && cbpb !=0) {		
dbquant	1-2	vlclbf
if (mb_type == '01' || mb_type == '1')		
motion_vector("forward")		
}		
for (i = 0; i < block_count; i++)		
if(!transparent_block(i))		
block(i)		
}		
}		
}		
}		
if(video_object_layer_shape=="grayscale" && !transparent_mb()) {		
for(j=0; j<aux_comp_count; j++) {		
if(vop_coding_type=="I" || ((vop_coding_type=="P" || (vop_coding_type=="S" && sprite_enable=="GMC")) && !not_coded && (derived_mb_type==3 || derived_mb_type==4))) {		
coda_i	1	bslbf
if(coda_i=="coded") {		
ac_pred_flag_alpha	1	bslbf
cbpa	1-6	vlclbf
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block())		
alpha_block(i)		
}		
} else { /* P, S(GMC) or B macroblock */		
if(vop_coding_type == "P" || (sprite_enable == "GMC" && (vop_coding_type=="S" || backward_reference_vop_coding_type=="S")) || co_located_not_coded != 1) {		
coda_pb	1-2	vlclbf
if(coda_pb=="coded") {		
cbpa	1-6	vlclbf
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block())		
alpha_block(i)		
}		
}		
}		
}		
}		

NOTE: The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4. backward_reference_vop_coding_type means the vop_coding_type of the backward reference VOP as described in subclause 7.6.7.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.6.1 MB Binary Shape Coding

mb_binary_shape_coding() {	No. of bits	Mnemonic
if(!(scalability && hierarchy_type == '0' && (enhancement_type == '0' || use_ref_shape == '0')) && !(scalability && video_object_layer_shape == "binary only")) {		
bab_type	1-7	vlclbf
if (vop_coding_type == 'P' || vop_coding_type == 'B' || (vop_coding_type == 'S' && sprite_enable == "GMC")) {		
if ((bab_type==1) || (bab_type == 6)) {		
mvds_x	1-18	vlclbf
mvds_y	1-18	vlclbf
}		
}		
if (bab_type >=4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlcbf
scan_type	1	bslbf
binary_arithmetic_code()		
}		
} else {		
if (!use_ref_shape || video_object_layer_shape == "binary only") {		
enh_bab_type	1-3	vlclbf
if (enh_bab_type == 3)		
scan_type	1	bslbf
if (enh_bab_type == 1 || enh_bab_type == 3)		
enh_binary_arithmetic_code()		
}		
}		
}		

backward_shape () {	No. of bits	Mnemonic
for(i=0; i<backward_shape_height/16; i++)		
for(j=0; j<backward_shape_width/16; j++) {		
bab_type	1-3	vlclbf
if (bab_type >=4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlcbf
scan_type	1	bslbf
binary_arithmetic_code()		
}		
}		
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

forward_shape () {	No. of bits	Mnemonic
for(i=0; i<forward_shape_height/16; i++)		
for(j=0; j<forward_shape_width/16; j++) {		
bab_type	1-3	vlclbf
if (bab_type >=4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlcbf
scan_type	1	bslbf
binary_arithmetic_code()		
}		
}		
}		

6.2.6.2 Motion vector

motion_vector (mode) {	No. of bits	Mnemonic
if (mode == „direct“) {		
horizontal_mv_data	1-13	vlclbf
vertical_mv_data	1-13	vlclbf
}		
else if (mode == „forward“) {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1) && (horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1) && (vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
else if (mode == „backward“) {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1) && (horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1) && (vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
}		

6.2.6.3 Interlaced Information

interlaced_information( ) {	No. of bits	Mnemonic
if ((derived_mb_type == 3) || (derived_mb_type == 4) || (cbp != 0))		
dct_type	1	bslbf

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (((vop_coding_type == “P”) &&
((derived_mb_type == 0) || (derived_mb_type == 1))) ||
((sprite_enable == “GMC”) && (vop_coding_type == “S”) &&
(derived_mb_type < 2) && (!mcsel)) ||
((vop_coding_type == “B”) && (mb_type != “1”))) {		

field_prediction	1	bslbf
if (field_prediction) {		

if (vop_coding_type == “P” || vop_coding_type == “S” ||
(vop_coding_type == “B” &&
mb_type != “001”)) {		

forward_top_field_reference	1	bslbf
forward_bottom_field_reference	1	bslbf
}		
if ((vop_coding_type == “B”) && (mb_type != “0001”)) {		
backward_top_field_reference	1	bslbf
backward_bottom_field_reference	1	bslbf
}		
}		
}		
}		

6.2.7 Block

The detailed syntax for the term “DCT coefficient” is fully described in clause 7.

block(i) {	**No. of bits**	**Mnemonic**
last = 0		
if(!data_partitioned && (derived_mb_type == 3 || derived_mb_type == 4)) {		
if(short_video_header == 1)		
intra_dc_coefficient	8	uimsbf
else if (use_intra_dc_vlc == 1) {		
if (i<4) {		
dct_dc_size_luminance	2-11	vlclbf
if(dct_dc_size_luminance != 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
} else {		
dct_dc_size_chrominance	2-12	vlclbf
if(dct_dc_size_chrominance !=0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
}		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (pattern_code[i])		
while (! last)		
DCT coefficient	3-24	vlclbf
}		
NOTE "last" is defined to be the LAST flag resulting from reading the most recent DCT coefficient.		

6.2.7.1 Alpha Block

The syntax for DCT coefficient decoding is the same as for block(i) in subclause 6.2.8.

alpha_block(i) {	**No. of bits**	**Mnemonic**
last = 0		

if(!data_partitioned &&
(vop_coding_type == "I" || (vop_coding_type == "P" ||
(vop_coding_type == "S" && sprite_enable == "GMC"))
&& !not_coded &&
(derived_mb_type == 3 || derived_mb_type == 4)))) {		

dct_dc_size_alpha	2-11	vlclbf
if(dct_dc_size_alpha != 0)		
dct_dc_differential	1-12	vlclbf
if (dct_dc_size_alpha > 8)		
marker_bit	1	bslbf
}		
if (pattern_code[i])		
while (! last)		
DCT coefficient	3-24	vlclbf
}		
NOTE "last" is defined to be the LAST flag resulting from reading the most recent DCT coefficient.		

6.2.8 Still Texture Object

StillTextureObject() {	**No. of bits**	**Mnemonic**
still_texture_object_start_code	32	bslbf
if (visual_object_verid != 0001) {		
tiling_disable	1	bslbf
texture_error_resilience_disable	1	bslbf
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf
texture_object_layer_shape	2	uimsbf
quantisation_type	2	uimsbf
if (quantisation_type == 2) {		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

spatial_scalability_levels	4	uimsbf
if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf
if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels – 1; i++)		
wavelet_layer_index	4	uimsbf
}		
}		
if (wavelet_download == “1”){		
uniform_wavelet_filter	1	uimsbf
if (uniform_wavelet_filter == “1”)		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters( )		
}		
wavelet_stuffing	3	uimsbf
if(!texture_error_resilience_disable) {		
target_segment_length	16	uimsbf
marker_bit	1	bslbf
}		
if(texture_object_layer_shape == “00”) {		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else if (texture_object_layer_shape == “01” or “11”) {		
horizontal_ref	15	uimsbf
marker_bit	1	bslbf
vertical_ref	15	uimsbf
marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
/* if tiling_disable == “1” and texture_object_layer_shape == “01” configuration information precedes this point; elementary stream data follows. See Annex K */		
if(tiling_disable == “1”)		
shape_object_decoding()		
}		
if (tiling_disable == “0”){		
tile_width	15	uimsbf
marker_bit	1	bslbf
tile_height	15	uimsbf
marker_bit	1	bslbf
number_of_tiles	16	uimsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

tiling_jump_table_enable	1	bslbf
if (tiling jump_table_enable == “1”) {		
for (i=0; i<number_of_tiles; i++) {		
tile_size_high	16	uimsbf
market_bit	1	bslbf
tile_size_low	16	uimsbf
market_bit	1	bslbf
}		
}		
next_start_code()		
}		
/* if tiling_disable == “0” or texture_object_layer_shape == “00” configuration information precedes this point; elementary stream data follows. See Annex K */		
do {		
if(tiling_disable == “0”) {		
texture_tile_start_code	32	bslbf
tile_id	16	uimsbf
if (texture_object_layer_shape == “01”) {		
marker_bit	1	bslbf
texture_tile_type	2	uimsbf
marker_bit	1	bslbf
}		
}		
if (!texture_error_resilience_disable) {		
if (texture_object_layer_shape== “01” && tiling_disable==“0”) {		
if (texture_tile_type==“boundary tile”)		
shape_object_decoding()		
}		
while (nextbit_bytealigned () == texture_marker) {		
TexturePacketHeader ()		
do {		
while (texture_unit_not_completed) {		
DecodeStu()		
if (segment_length >= target_segment_length)		
decode_segment_marker()		
}		
} while (nextbit_bytealigned () != texture_marker)		
}		
}		
else {		
if (texture_object_layer_shape!= “01” && tiling_disable==“0”) {		
if (texture_tile_type==“boundary tile”)		
shape_object_decoding()		
}		
for (color = “y”, “u”, “v” ,”α”) {		
wavelet_dc_decode()		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (quantisation_type == 1) {		
TextureLayerSQ ( )		
}		
else if (quantisation_type == 2) {		
if (start_code_enable == 1) {		
do {		
TextureSpatialLayerMQ()		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i =0; i<spatial_scalability_levels; i++)		
TextureSpatialLayerMQNSC()		
}		
}		
else if (quantisation_type == 3) {		
for (color = “y”, “u”, “v”)		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
max_bitplanes	5	uimsbf
if (scan_direction == 0) {		
do {		
TextureSNRLayerBQ ( )		
} while (next_bits() == texture_snr_layer_start_code)		
} else {		
do {		
TextureSpatialLayerBQ ( )		
} while (next_bits() == texture_spatial_layer_start_code)		
}		
}		
} /* error_resi_disable */		
if (tiling_disable == “0”)		
next_start_code()		
} while (nextbits_bytealigned () == texture_tile_start_code)		
}		
else { /* version 1 */		
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf
texture_object_layer_shape	2	uimsbf
quantisation_type	2	uimsbf
if (quantisation_type == 2) {		
spatial_scalability_levels	4	uimsbf

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf
if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels – 1; i++)		
wavelet_layer_index	4	
}		
}		
if (wavelet_download == “1”){		
uniform_wavelet_filter	1	uimsbf
if (uniform_wavelet_filter == “1”)		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters()		
}		
wavelet_stuffing	3	uimsbf
if(texture_object_layer_shape == “00”){		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else {		
horizontal_ref	15	imsbf
marker_bit	1	bslbf
vertical_ref	15	imsbf
marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
shape_object_decoding ( )		
}		
/* configuration information precedes this point; elementary stream data follows. See Annex K */		
for (color = “y”, “u”, “v”)		
wavelet_dc_decode()		
if(quantisation_type == 1)		
TextureLayerSQ()		
else if (quantisation_type == 2) {		
if (start_code_enable == 1) {		
do {		
TextureSpatialLayerMQ()		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i =0; i<spatial_scalability_levels; i++)		
TextureSpatialLayerMQNSC()		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}		
else if (quantisation_type == 3) {		
for (color = “y”, “u”, “v” ,”α”)		
do{		
quant_byte		
} while(quant_byte >>7)		
max_bitplanes		
if (scan_direction == 0) {		
do {		
TextureSNRLayerBQ()		
} while (next_bits() == texture_snr_layer_start_code)		
} else {		
do {		
TextureSpatialLayerBQ()		
} while (next_bits() == texture_spatial_layer_start_code)		
}		
}		
}		
}		

NOTE 1: The value of texture_unit_not_completed is '0' if the decoding of one sub-unit in a texture unit is completed. Otherwise the value is '1'.

NOTE 2: The value of first_packet_decoded is set to ‘0’ if the first packet has not been decoded. The value of first_packet_decoded is set to ‘1’ after the first packet has been decoded.

TexturePacketHeader() {	**No. of bits**	**Mnemonic**
next_texture_marker()		
texture_marker	17	bslbf
do {		
TU_first	8	uimsbf
} while (TU_first >> 7)		
do{		
TU_last	8	uimsbf
} while (TU_last >> 7)		
header_extention_code	1	bslbf
if (header_extention_code) {		
texture_object_id	16	uimsbf
marker_bit	1	bslbf
wavelet_filter_type	1	uimsbf
wavelet_download	1	uimsbf
wavelet_decomposition_levels	4	uimsbf
scan_direction	1	bslbf
start_code_enable	1	bslbf
texture_object_layer_shape	2	uimsbf
quantisation_type	2	uimsbf
if (quantisation_type == 2) {		
spatial_scalability_levels	4	uimsbf
if (spatial_scalability_levels != wavelet_decomposition_levels) {		
use_default_spatial_scalability	1	uimsbf

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

if (use_default_spatial_layer_size == 0)		
for (i=0; i<spatial_scalability_levels – 1; i++)		
wavelet_layer_index	4	uimsbf
}		
}		
if (wavelet_download == “1”){		
uniform_wavelet_filter	1	uimsbf
if (uniform_wavelet_filter == “1”)		
download_wavelet_filters()		
else		
for (i=0; i<wavelet_decomposition_levels; i++)		
download_wavelet_filters( )		
}		
wavelet_stuffing	3	uimsbf
if(texture_object_layer_shape == “00”) {		
texture_object_layer_width	15	uimsbf
marker_bit	1	bslbf
texture_object_layer_height	15	uimsbf
marker_bit	1	bslbf
}		
else {		
if (!first_packet_decoded) {		
horizontal_ref	15	uimsbf
marker_bit	1	bslbf
vertical_ref	15	uimsbf
marker_bit	1	bslbf
object_width	15	uimsbf
marker_bit	1	bslbf
object_height	15	uimsbf
marker_bit	1	bslbf
}		
}		
if (tiling_disable == “0”){		
tile_width	15	uimsbf
marker_bit	1	bslbf
tile_height	15	uimsbf
marker_bit	1	bslbf
}		
target_segment_length	16	uimsbf
marker_bit	1	bslbf
}		
}		

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

DecodeStu() {	No. of bits	Mnemonic
for (color = “y”, “u”, “v” ,”α”)		
wavelet_dc_decode ()		
if(quantisation_type == 1) {		
TextureLayerSQ()		
}		
else if (quantisation_type == 2) {		
if (start_code_enable == 1) {		
do {		
TextureSpatialLayerMQ()		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i =0; i<spatial_scalability_levels; i++)		
TextureSpatialLayerMQNSC()		
}		
}		
else if (quantisation_type == 3) {		
for (color = “y”, “u”, “v” ,”α”)		
do{		
quant_byte	8	uimsbf
} while(quant_byte >> 7)		
max_bitplanes	5	uimsbf
if (scan_direction == 0) {		
do {		
TextureSNRLayerBQ()		
} while (next_bits() == texture_snr_layer_start_code)		
} else {		
do {		
TextureSpatialLayerBQ()		
} while (next_bits() == texture_spatial_layer_start_code)		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.8.1 TextureLayerSQ

TextureLayerSQ() {	No. of bits	Mnemonic
if (scan_direction == 0) {		
for (“y”, “u”, “v”,”α”) {		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
for (i=0; i<wavelet_decomposition_levels; i++)		
if (i!=0 || color!= “u“,“v“) {		
max_bitplane[i]	5	uimsbf
if ((i+1)%4==0)		
marker_bit	1	bslbf
}		
}		
for (i = 0; i<tree_blocks; i++)		
for (color = “y”, “u”, “v” ,”α”)		
arith_decode_highbands_td()		
} else {		
if (start_code_enable) {		
do {		
TextureSpatialLayerSQ()		
} while (next_bits() == texture_spatial_layer_start_code)		
} else {		
for (i = 0; i< wavelet_decomposition_levels; i++)		
TextureSpatialLayerSQNSC()		
}		
}		
}		
NOTE The value of tree_block is that wavelet coefficients are organized in a tree structure which is rooted in the low-low band (DC band) of the wavelet decomposition, then extends into the higher frequency bands at the same spatial location. Note the DC band is encoded separately.		

6.2.8.2 TextureSpatialLayerSQ

TextureSpatialLayerSQ() {	No. of bits	Mnemonic
texture_spatial_layer_start_code	32	bslbf
texture_spatial_layer_id	5	uimsbf
TextureSpatialLayerSQNSC()		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.8.3 TextureSpatialLayerSQNSC

TextureSpatialLayerSQNSC() {	No. of bits	Mnemonic
for (color="y","u","v","α") {		
if ((first_wavelet_layer && color=="y","α") || (second_wavelet_layer && color=="u","v"))		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
if (color =="y","α")		
max_bitplanes	5	uimbsf
else if (!first_wavelet_layer)		
max_bitplanes	5	uimbsf
}		
for (color="y","u","v","α")		
if (color="y","α" || !first_wavelet_layer)		
arith_decode_highbands_bb()		
}		
NOTE The value of first_wavelet_layer becomes "true" when the variable 'i' of subclause 6.2.8.1 TextureLayerSQ() equals to zero. Otherwise, it is "false". The value of second_wavelet_layer becomes "true" when the variable 'i' of sublause 6.2.8.1 TextureLayerSQ() equals to one. Otherwise, it is "false".		

6.2.8.4 TextureSpatialLayerMQ

TextureSpatialLayerMQ() {	No. of bits	Mnemonic
texture_spatial_layer_start_code	32	bslbf
texture_spatial_layer_id	5	uimsbf
snr_scalability_levels	5	uimsbf
do {		
TextureSNRLayerMQ( )		
} while (next_bits() == texture_snr_layer_start_code)		
}		

6.2.8.5 TextureSpatialLayerMQNSC

TextureSpatialLayerMQNSC() {	No. of bits	Mnemonic
snr_scalability_levels	5	uimsbf
for (i =0; i<snr_scalability_levels; i++)		
TextureSNRLayerMQNSC ( )		
}		

6.2.8.6 TextureSNRLayerMQ

TextureSNRLayerMQ(){		
texture_snr_layer_start_code	32	bslbf
texture_snr_layer_id	5	uimsbf
TextureSNRLayerMQNSC()		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.8.7 TextureSNRLayerMQNSC

TextureSNRLayerMQNSC(){	No. of bits	Mnemonic
if (spatial_scalability_levels == wavelet_decomposition_levels && spatial_layer_id == 0) {		
for (color = “y“,”α”) {		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
for (i=0; i<spatial_layers; i++) {		
max_bitplane[i]	5	uimsbf
if ((i+1)%4 == 0)		
marker_bit	1	bslbf
}		
}		
}		
else {		
for (color=“y”, “u”, “v” ,”α”) {		
do {		
quant_byte	8	uimsbf
} while (quant_byte >> 7)		
for (i=0; i<spatial_layers; i++) {		
max_bitplane[i]	5	uimsbf
if ((i+1)%4 == 0)		
marker_bit	1	bslbf
}		
}		
}		
if (scan_direction == 0) {		
for (i = 0; i<tree_blocks; i++)		
for (color = “y”, “u”, “v” ,”α”)		
if (wavelet_decomposition_layer_id != 0 || color != “u”, “v”)		
arith_decode_highbands_td()		
} else {		
for (i = 0; i< spatial_layers; i++) {		
for (color = “y”, “u”, “v” ,”α”) {		
if (wavelet_decomposition_layer_id != 0 || color != “u”, “v”)		
arith_decode_highbands_bb()		
}		
}		
}		
}		
NOTE The value of spatial_layers is equivalent to the maximum number of the wavelet decomposition layers in that scalability layer.		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.8.8 TextureSpatialLayerBQ

TextureSpatialLayerBQ() {	No. of bits	Mnemonic
texture_spatial_layer_start_code	32	bslbf
texture_spatial_layer_id	5	uimsbf
for (i=0; i<max_bitplanes; i++) {		
texture_snr_layer_start_code	32	bslbf
texture_snr_layer_id	5	uimsbf
TextureBitPlaneBQ()		
next_start_code()		
}		
}		

6.2.8.9 TextureBitPlaneBQ

TextureBitPlaneBQ () {	No. of bits	Mnemonic
for (color = “y”, “u”, “v” ,”α”)		
if (wavelet_decomposition_layer_id == 0){		
all_nonzero[color]	1	bslbf
if (all_nonzero[color] == 0) {		
all_zero[color]	1	bslbf
if (all_zero[color]==0) {		
lh_zero[color]	1	bslbf
hl_zero[color]	1	bslbf
hh_zero[color]	1	bslbf
}		
}		
}		
if (wavelet_decomposition_layer_id != 0 ||color != “u”, “v”){		
if(all_nonzero[color]==1 || all_zero[color]==0){		
if (scan_direction == 0)		
arith_decode_highbands_bilevel_bb()		
else		
arith_decode_highbands_bilevel_td()		
}		
}		
}		
}		

6.2.8.10 TextureSNRLayerBQ

TextureSNRLayerBQ() {	No. of bits	Mnemonic
texture_snr_layer_start_code	32	bslbf
texture_snr_layer_id	5	uimsbf
for (i=0; i<wavelet_decomposition_levels; i++) {		
texture_spatial_layer_start_code	32	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

texture_spatial_layer_id	5	uimsbf
TextureBitPlaneBQ()		
next_start_code ( )		
}		
}		

6.2.8.11 DownloadWaveletFilters

download_wavelet_filters() {	**No. of bits**	**Mnemonic**
lowpass_filter_length	4	uimsbf
highpass_filter_length	4	uimsbf
do{		
if (wavelet_filter_type == 0) {		
filter_tap_integer	16	imsbf
marker_bit	1	bslbf
} else {		
filter_tap_float_high	16	uimsbf
marker_bit	1	bslbf
filter_tap_float_low	16	uimsbf
marker_bit	1	bslbf
}		
} while (lowpass_filter_length--)		
do{		
if (wavelet_filter_type == 0){		
filter_tap_integer	16	imsbf
marker_bit	1	bslbf
} else {		
filter_tap_float_high	16	uimsbf
marker_bit	1	bslbf
filter_tap_float_low	16	uimsbf
marker_bit	1	bslbf
}		
} while (highpass_filter_length--)		
if (wavelet_filter_type == 0) {		
integer_scale	16	uimsbf
marker_bit	1	bslbf
}		
}		

6.2.8.12 Wavelet dc decode

wavelet_dc_decode() {	**No. of bits**	**Mnemonic**
mean	8	uimsbf
do{		
quant_dc_byte	8	uimsbf
} while(quant_dc_byte >>7)		

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

do{		
band_offset_byte	8	uimsbf
} while (band_offset_byte >>7)		
do{		
band_max_byte	8	uimsbf
} while (band_max_byte >>7)		
arith_decode_dc()		
}		

6.2.8.13 Wavelet higher bands decode

wavelet_ higher_bands_decode() {	**No. of bits**	**Mnemonic**
do{		
root_max_alphabet_byte	8	uimsbf
} while (root_max_alphabet_byte >>7)		
marker_bit	1	bslbf
do{		
valz_max_alphabet_byte	8	uimsbf
} while (valz_max_alphabet_byte >>7)		
do{		
valnz_max_alphabet_byte	8	uimsbf
} while (valnz_max_alphabet_byte >>7)		
arith_decode_highbands()		
}		

6.2.8.14 Shape Object Decoding

shape_object_decoding() {	**No. of bits**	**Mnemonic**
change_conv_ratio_disable	1	bslbf
sto_constant_alpha	1	bslbf
if (sto_constant_alpha)		
sto_constant_alpha_value	8	bslbf
if (visual_object_verid != 0001) {		
marker_bit	1	bslbf
for(i=0; i<shape_base_layer_height_blocks(); i++) {		
for(j=0; j<shape_base_layer_width_blocks(); j++) {		
bab_type	1-2	vlclbf
if (bab_type == 4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlclbf
scan_type	1	bslbf
binary_arithmetic_decode()		
}		
}		
}		
marker_bit	1	bslbf
if (!start_code_enable) {		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sto_shape_coded_layers	4	uimsbf
marker_bit	1	bslbf
for(k = 0; k < sto_shape_coded_layers; k++) {		
for(i=0;i<shape_enhanced_layer_height_blocks(); i++)		
for(j=0; j<shape_enhanced_layer_width_blocks(); j++)		
enh_binary_arithmetic_decode()		
marker_bit	1	bslbf
}		
}		
else {		
next_start_code()		
while (nextbits() == texture_shape_layer_start_code) {		
texture_shape_layer_start_code	32	bslbf
texture_shape_layer_id	5	uimsbf
marker_bit	1	bslbf
for(i=0;i<shape_enhanced_layer_height_blocks(); i++)		
for(j=0; j<shape_enhanced_layer_width_blocks(); j++)		
enh_binary_arithmetic_decode()		
marker_bit	1	bslbf
next_start_code()		
}		
texture_spatial_layer_start_code	32	bslbf
texture_spatial_layer_id	5	uimsbf
marker_bit	1	bslbf
}		
} else { /* version 1 */		
for (i=0 ; i<((object_width+15)/16)*(object_height+15)/16) ; i++) {		
bab_type	1-2	vlclbf
if (bab_type==4) {		
if (!change_conv_ratio_disable)		
conv_ratio	1-2	vlclbf
scan_type	1	bslbf
binary_arithmetic_decode()		
}		
}		
}		

6.2.9 Mesh Object

MeshObject() {	**No. of bits**	**Mnemonic**
mesh_object_start_code	32	bslbf
do{		
MeshObjectPlane()		
} while (next_bits_bytealigned() == mesh_object_plane_start_code || next_bits_bytealigned() != '0000 0000 0000 0000 0000 0001')		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.9.1 Mesh Object Plane

MeshObjectPlane() {	No. of bits	Mnemonic
MeshObjectPlaneHeader()		
MeshObjectPlaneData()		
}		

MeshObjectPlaneHeader() {	No. of bits	Mnemonic
if (next_bits_bytealigned()=='0000 0000 0000 0000 0000 0001'){		
next_start_code()		
mesh_object_plane_start_code	32	bslbf
}		
is_intra	1	bslbf
mesh_mask	1	bslbf
temporal_header()		
}		

MeshObjectPlaneData() {	No. of bits	Mnemonic
if (mesh_mask == 1) {		
if (is_intra == 1)		
mesh_geometry()		
else		
mesh_motion()		
}		
}		

6.2.9.2 Mesh geometry

mesh_geometry() {	No. of bits	Mnemonic
mesh_type _code	2	bslbf
if (mesh_type_code == '01') {		
nr_of_mesh_nodes_hor	10	uimsbf
nr_of_mesh_nodes_vert	10	uimsbf
marker_bit	1	uimsbf
mesh_rect_size_hor	8	uimsbf
mesh_rect_size_vert	8	uimsbf
triangle_split_code	2	bslbf
}		
else if (mesh_type_code == '10') {		
nr_of_mesh_nodes	16	uimsbf
marker_bit	1	uimsbf
nr_of_boundary_nodes	10	uimsbf
marker_bit	1	uimsbf
node0_x	13	simsbf
marker_bit	1	uimsbf

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

node0_y	13	simsbf
marker_bit	1	uimsbf
for (n=1; n < nr_of_mesh_nodes; n++) {		
delta_x_len_vlc	2-12	vlclbf
if (delta_x_len_vlc)		
delta_x	1-14	vlclbf
delta_y_len_vlc	2-12	vlclbf
if (delta_y_len_vlc)		
delta_y	1-14	vlclbf
}		
}		
}		

6.2.9.3 Mesh motion

mesh_motion() {	**No. of bits**	**Mnemonic**
motion_range_code	3	bslbf
for (n=0; n <nr_of_mesh_nodes; n++) {		
node_motion_vector_flag	1	bslbf
if (node_motion_vector_flag == '0') {		
delta_mv_x_vlc	1-13	vlclbf
if ((motion_range_code != 1) && (delta_mv_x_vlc != 0))		
delta_mv_x_res	1-6	uimsbf
delta_mv_y_vlc	1-13	vlclbf
if ((motion_range_code != 1) && (delta_mv_y_vlc != 0))		
delta_mv_y_res	1-6	uimsbf
}		
}		
}		

6.2.10 FBA Object

fba_object() {	**No. of bits**	**Mnemonic**
fba_object_start_code	32	bslbf
do {		
fba_object_plane()		
} while (!((nextbits_bytealigned() == '000 0000 0000 0000 0000 0000') && (nextbits_bytealigned() != fba_object_plane_start_code)))		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.10.1 FBA Object Plane

fba_object_plane() {	No. of bits	Mnemonic
fba_object_plane_header()		
fba_object_plane_data()		
}		

fba_object_plane_header() {	No. of bits	Mnemonic
If (nextbits_bytealigned() == '0000 0000 0000 0000 0000 0001'){		
next_start_code()		
fba_object_plane_start_code	32	bslbf
}		
is_intra	1	bslbf
fba_object_mask	2	bslbf
temporal_header()		
}		

fba_object_plane_data() {	No. of bits	Mnemonic
if(fba_object_mask &'01') {		
if(is_intra) {		
fap_quant	5	uimsbf
for (group_number = 1; group_number <= 10; group_number++) {		
marker_bit	1	uimsbf
fap_mask_type	2	bslbf
if(fap_mask_type == '01'|| fap_mask_type == '10')		
fap_group_mask[group_number]	2-16	vlcbf
}		
fba_suggested_gender	1	bslbf
fba_object_coding_type	1	bslbf
if(fba_object_coding_type == 0) {		
is_i_new_max	1	bslbf
is_i_new_min	1	bslbf
is_p_new_max	1	bslbf
is_p_new_min	1	bslbf
decode_new_minmax()		
decode_ifap()		
}		
if(fba_object_coding_type == 1)		
decode_i_segment()		
}		
else {		
if(fba_object_coding_type == 0)		
decode_pfap()		
if(fba_object_coding_type == 1)		
decode_p_segment()		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}		
}		
if(fba_object_mask &'10') {		
if(is_intra) {		
bap_pred_quant_index	5	uimsbf
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
marker_bit	1	uimsbf
bap_mask_type	2	bslbf
if(bap_mask_type == '01')		
bap_group_mask[group_number]	3-22	vlcbf
else if (bap_mask_type == '00') {		
for(i=0; i<BAPS_IN_GROUP[group_number];i++) {		
bap_group_mask[group_mask][i] = 0		
}		
}		
else if (bap_mask_type == '11') {		
for(i=0; i<BAPS_IN_GROUP[group_number];i++) {		
bap_group_mask[group_mask][i] = 1		
}		
}		
}		
fba_suggested_gender	1	bslbf
fba_object_coding_type	1	bslbf
if (fba_object_coding_type == 0) {		
bap_is_i_new_max	1	bslbf
bap_is_i_new_min	1	bslbf
bap_is_p_new_max	1	bslbf
bap_is_p_new_min	1	bslbf
decode_bap_new_minmax()		
decode_ibap()		
}		
if(fba_object_coding_type == 1)		
decode_bap_i_segment()		
}		
else {		
if (fba_object_coding_type == 0)		
decode_pbap()		
if(fba_object_coding_type == 1)		
decode_bap_p_segment()		
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

temporal_header() {	No. of bits	Mnemonic
if (is_intra) {		
is_frame_rate	1	bslbf
if(is_frame_rate)		
decode_frame_rate()		
is_time_code	1	bslbf
if (is_time_code)		
time_code	18	bslbf
}		
skip_frames	1	bslbf
if(skip_frames)		
decode_skip_frames()		
}		

6.2.10.2 Decode frame rate and skip frames

decode_frame_rate(){	No. of bits	Mnemonic
frame_rate	8	uimsbf
seconds	4	uimsbf
frequency_offset	1	uimsbf
}		

decode_skip_frames(){	No. of bits	Mnemonic
do{		
number_of_frames_to_skip	4	uimsbf
} while (number_of_frames_to_skip = “1111”)		
}		

6.2.10.3 Decode new minmax

decode_new_minmax() {	No. of bits	Mnemonic
if (is_i_new_max) {		
for (group_number = 2, j=0, group_number <= 10, group_number++)		
for (i=0; i < NFAP[group_number]; i++, j++) {		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		
i_new_max[j]	5	uimsbf
}		
if (is_i_new_min) {		
for (group_number = 2, j=0, group_number <= 10, group_number++)		
for (i=0; i < NFAP[group_number]; i++, j++) {		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

i_new_min[j]	5	uimsbf
}		
if (is_p_new_max) {		
for (group_number = 2, j=0, group_number <= 10, group_number++)		
for (i=0; i < NFAP[group_number]; i++, j++) {		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		
p_new_max[j]	5	uimsbf
}		
if (is_p_new_min) {		
for (group_number = 2, j=0, group_number <= 10, group_number++)		
for (i=0; i < NFAP[group_number]; i++, j++) {		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (fap_group_mask[group_number] & (1 <<i))		
p_new_min[j]	5	uimsbf
}		
}		
}		

6.2.10.4 Decode ifap

decode_ifap(){	**No. of bits**	**Mnemonic**
for (group_number = 1, j=0; group_number <= 10; group_number++) {		
if (group_number == 1) {		
if(fap_group_mask[1] & 0x1)		
decode_viseme()		
if(fap_group_mask[1] & 0x2)		
decode_expression()		
} else {		
for (i= 0; i<NFAP[group_number]; i++, j++) {		
if(fap_group_mask[group_number] & (1 << i)) {		
aa_decode(ifap_Q[j],ifap_cum_freq[j])		
}		
}		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.10.5 Decode pfap

decode_pfap(){	No. of bits	Mnemonic
for (group_number = 1, j=0; group_number <= 10; group_number++) {		
if (group_number == 1) {		
if(fap_group_mask[1] & 0x1)		
decode_viseme()		
if(fap_group_mask[1] & 0x2)		
decode_expression()		
} else {		
for (i= 0; i<NFAP[group_number]; i++, j++) {		
if(fap_group_mask[group_number] & (1 << i)) {		
aa_decode(pfap_diff[j], pfap_cum_freq[j])		
}		
}		
}		
}		
}		

6.2.10.6 Decode viseme and expression

decode_viseme() {	No. of bits	Mnemonic
aa_decode(viseme_select1Q, viseme_select1_cum_freq)		vlclbf
aa_decode(viseme_select2Q, viseme_select2_cum_freq)		vlclbf
aa_decode(viseme_blendQ, viseme_blend_cum_freq)		vlclbf
viseme_def	1	bslbf
}		

decode_expression() {	No. of bits	Mnemonic
aa_decode(expression_select1Q, expression_select1_cum_freq)		vlclbf
aa_decode(expression_intensity1Q, expression_intensity1_cum_freq)		vlclbf
aa_decode(expression_select2Q, expression_select2_cum_freq)		vlclbf
aa_decode(expression_intensity2Q, expression_intensity2_cum_freq)		vlclbf
init_face	1	bslbf
expression_def	1	bslbf
}		

6.2.10.7 Decode i_segment

decode_i_segment(){	No. of bits	Mnemonic
for (group_number= 1, j=0; group_number<= 10; group_number++) {		
if (group_number == 1) {		
if(fap_group_mask[1] & 0x1)		
decode_i_viseme_segment()		
if(fap_group_mask[1] & 0x2)		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

decode_i_expression_segment()		
} else {		
for(i=0; i<NFAP[group_number]; i++, j++) {		
if(fap_group_mask[group_number] & (1 << i)) {		
decode_i_dc(dc_Q[j])		
decode_ac(ac_Q[j])		
}		
}		
}		
}		
}		

6.2.10.8 Decode p_segment

decode_p_segment(){	**No. of bits**	**Mnemonic**
for (group_number = 1, j=0; group_number <= 10; group_number++) {		
if (group_number == 1) {		
if(fap_group_mask[1] & 0x1)		
decode_p_viseme_segment()		
if(fap_group_mask[1] & 0x2)		
decode_p_expression_segment()		
} else {		
for (i=0; i<NFAP[group_number]; i++, j++) {		
If(fap_group_mask[group_number] & (1 << i)) {		
decode_p_dc(dc_Q[j])		
decode_ac(ac_Q[j])		
}		
}		
}		
}		
}		

6.2.10.9 Decode viseme and expression

decode_i_viseme_segment(){	**No. of bits**	**Mnemonic**
viseme_segment_select1q[0]	4	uimsbf
viseme_segment_select2q[0]	4	uimsbf
viseme_segment_blendq[0]	6	uimsbf
viseme_segment_def[0]	1	bslbf
for (k=1; k<16, k++) {		
viseme_segment_select1q_diff[k]		vlclbf
viseme_segment_select2q_diff[k]		vlclbf
viseme_segment_blendq_diff[k]		vlclbf
viseme_segment_def[k]	1	bslbf
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

decode_p _viseme_segment(){	No. of bits	Mnemonic
for (k=0; k<16, k++) {		
viseme_segment_select1q_diff[k]		vlclbf
viseme_segment_select2q_diff[k]		vlclbf
viseme_segment_blendq_diff[k]		vlclbf
viseme_segment_def[k]	1	bslbf
}		
}		

decode_i_expression_segment(){	No. of bits	Mnemonic
expression_segment_select1q[0]	4	uimsbf
expression_segment_select2q[0]	4	uimsbf
expression_segment_intensity1q[0]	6	uimsbf
expression_segment_intensity2q[0]	6	uimsbf
expression_segment_init_face[0]	1	bslbf
expression_segment_def[0]	1	bslbf
for (k=1; k<16, k++) {		
expression_segment_select1q_diff[k]		vlclbf
expression_segment_select2q_diff[k]		vlclbf
expression_segment_intensity1q_diff[k]		vlclbf
expression_segment_intensity2q_diff[k]		vlclbf
expression_segment_init_face[k]	1	bslbf
expression_segment_def[k]	1	bslbf
}		
}		

decode_p _expression_segment(){	No. of bits	Mnemonic
for (k=0; k<16, k++) {		
expression_segment_select1q_diff[k]		vlclbf
expression_segment_select2q_diff[k]		vlclbf
expression_segment_intensity1q_diff[k]		vlclbf
expression_segment_intensity2q_diff[k]		vlclbf
expression_segment_init_face[k]	1	bslbf
expression_segment_def[k]	1	bslbf
}		
}		

decode_i_dc(dc_q) {	No. of bits	Mnemonic
dc_q	16	simsbf
if(dc_q == -256*128)		
dc_q	31	simsbf
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

decode_p_dc(dc_q_diff) {	No. of bits	Mnemonic
dc_q_diff		vlclbf
dc_q_diff = dc_q_diff- 256		
if(dc_q_diff == -256)		
dc_q_diff	16	simsbf
if(dc_Q == 0-256*128)		
dc_q_diff	32	simsbf
}		

decode_ac(ac_Q[i]) {	No. of bits	Mnemonic
this = 0		
next = 0		
while(next < 15) {		
count_of_runs		vlclbf
if (count_of_runs == 15)		
next = 16		
else {		
next = this+1+count_of_runs		
for (n=this+1; n<next; n++)		
ac_q[i][n] = 0		
ac_q[i][next]		vlclbf
if(ac_q[i][next] == 256)		
decode_i_dc(ac_q[i][next])		
else		
ac_q[i][next] = ac_q[i][next]-256		
this = next		
}		
}		
}		

6.2.10.10 Decode bap min max

decode_bap_new_minmax() {	No. of bits	Mnemonic
if (bap_is_i_new_max) {		
for (group_number=1;group_number<= BAP_NUM_GROUPS; group_number++)		
for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_i_new_max[j]	5	uimsbf
}		
if (bap_is_i_new_min) {		
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++)		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_i_new_min[j]	5	uimsbf
}		
if (bap_is_p_new_max) {		
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++)		
for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_p_new_max[j]	5	uimsbf
}		
if (bap_is_p_new_min) {		
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++)		
for (i=0; i < NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if (!(i & 0x3))		
marker_bit	1	uimsbf
if (bap_group_mask[group_number] & (1 <<i))		
bap_p_new_min[j]	5	uimsbf
}		
}		
}		

6.2.10.11 Decode ibap

decode_ibap(){	**No. of bits**	**Mnemonic**
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
for (i= 0; i<NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if(bap_group_mask[group_number] & (1 << i)) {		
aa_decode(ibap_Q[j],ibap_cum_freq[j])		
}		
}		
}		
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

6.2.10.12 Decode pbap

decode_pbap() {	No. of bits	Mnemonic
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
for (i= 0; i<NBAP_GROUP[group_number]; i++) {		
j=BAPS_IN_GROUP[group_number][i]		
if(bap_group_mask[group_number] & (1 << i)) {		
aa_decode(pbap_diff[j], pbap_cum_freq[j])		
}		
}		
}		
}		

6.2.10.13 Decode bap i segment

decode_bap_i_segment(){	No. of bits	Mnemonic
for (group_number= 1; group_number<= BAP_NUM_GROUPS; group_number++) {		
for(i=0; i<NBAP_GROUP[group_number]; i++) {		
if(bap_group_mask[group_number] & (1 << i)) {		
j=BAPS_IN_GROUP[group_number][i]		
decode_i_dc(dc_Q[j])		
decode_ac(ac_Q[j])		
}		
}		
}		
}		

6.2.10.14 Decode bap p segment

decode_bap_p_segment(){	No. of bits	Mnemonic
for (group_number = 1; group_number <= BAP_NUM_GROUPS; group_number++) {		
for (i=0; i<NBAP_GROUP[group_number]; i++) {		
If(bap_group_mask[group_number] & (1 << i)) {		
j = BAPS_IN_GROUP[group_number][i]		
decode_p_dc(dc_Q[j])		
decode_ac(ac_Q[j])		
}		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.11 3D Mesh Object

6.2.11.1 3D_Mesh_Object

3D_Mesh_Object () {	No. of bits	Mnemonic
3D_MO_start_code	16	uimsbf
3D_Mesh_Object_Header()		
do {		
3D_Mesh_Object_Layer()		
} while (nextbits_bytealigned() == 3D_MOL_start_code)		
}		

6.2.11.2 3D_Mesh_Object_Header

3D_Mesh_Object_Header() {	No. of bits	Mnemonic
ccw	1	bslbf
convex	1	bslbf
solid	1	bslbf
creaseAngle	6	**uimsbf**
coord_header()		
normal_header()		
color_header()		
texCoord_header()		
ce_SNHC	1	**bslbf**
if (ce_SNHC == '1')		
ce_SNHC _header()		
}		

6.2.11.3 3D_Mesh_Object_Layer

3D_Mesh_Object_Layer () {	No. of bits	Mnemonic
3D_MOL_start_code	16	uimsbf
mol_id	8	uimsbf
if (ce_SNHC == '1') {		
ce_SNHC _n_vertices	24	uimsbf
ce_SNHC _n_triangles	24	uimsbf
ce_SNHC _n_edges	24	uimsbf
}		
if (mol_id == 0)		
3DMeshObject_Base_Layer()		
else		
3DMeshObject_Refinement_Layer()		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.11.4 3D_Mesh_Object_Base_Layer

3DmeshObject_Base_Layer()	No. of bits	Mnemonic
do {		
3D_MOBL_start_code	16	uimsbf
mobl_id	8	uimsbf
while (!bytealigned())		
one_bit	1	bslbf
qf_start()		
if (3D_MOBL_start_code == “partition_type_0”) {		
do {		
connected_component()		
qf_decode(**last_component**, last_component_context)		vlclbf
} while (last_component == ‘0’)		
}		
else if (3D_MOBL_start_code == “partition_type_1”) {		
vg_number=0		
do {		
vertex_graph()		
vg_number++		
qf_decode(**has_stitches**, has_stitches_context)		vlclbf
if (has_stitches == ‘1’)		
stitches()		
qf_decode(**codap_last_vg**, codap_last_vg_context)		vlclbf
} while (codap_last_vg == ‘0’)		
}		
else if (3D_MOBL_start_code == “partition_type_2”) {		
if(vg_number > 1)		
qf_decode(**codap_vg_id**)		vlclbf
qf_decode(**codap_left_bloop_idx**)		vlclbf
qf_decode(**codap_right_bloop_idx**)		vlclbf
qf_decode(**codap_bdry_pred**)		vlclbf
triangle_tree()		
triangle_data()		
}		
} while (nextbits_bytealigned() == 3D_MOBL_start_code)		
}		

6.2.11.5 coord_header

coord_header() {	No. of bits	Mnemonic
coord_binding	2	uimsbf
coord_bbox	1	bslbf
if (coord_bbox == ‘1’) {		
coord_xmin	32	bslbf
coord_ymin	32	bslbf
coord_zmin	32	bslbf
coord_size	32	bslbf
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

coord_quant	5	uimsbf
coord_pred_type	2	uimsbf
if (coord_pred_type==”tree_prediction” || coord_pred_type==’”parallelogram_prediction”) {		
coord_nlambda	2	uimsbf
for (i=1; i<coord_nlambda; i++)		
coord_lambda	4-27	simsbf
}		
}		

6.2.11.6 normal_header

normal_header() {	**No. of bits**	**Mnemonic**
normal_binding	2	uimsbf
if (normal_binding != “not_bound”) {		
normal_bbox	1	bslbf
normal_quant	5	uimsbf
normal_pred_type	2	uimsbf
if (normal_pred_type==”tree_prediction”|| normal_pred_type==”parallelogram_prediction”) {		
normal_nlambda	2	uimsbf
for (i=1; i<normal_nlambda; i++)		
normal_lambda	3-17	simsbf
}		
}		
}		

6.2.11.7 color_header

color_header() {	**No. of bits**	**Mnemonic**
color_binding	2	uimsbf
if (color_binding != “not_bound”) {		
color_bbox	1	bslbf
if (color_bbox == ‘1’) {		
color_rmin	32	bslbf
color_gmin	32	bslbf
color_bmin	32	bslbf
color_size	32	bslbf
}		
color_quant	5	uimsbf
color_pred_type	2	uimsbf
if (color_pred_type==”tree_prediction” || color_pred_type==’”parallelogram_prediction”) {		
color_nlambda	2	uimsbf
for (i=1; i<color_nlambda; i++)		
color_lambda	4-19	simsbf
}		
}		
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

6.2.11.8 texCoord_header

texCoord_header() {	No. of bits	Mnemonic
texCoord_binding	2	uimsbf
if (texCoord_binding != “not_bound”) {		
texCoord_bbox	1	bslbf
if (texCoord_bbox == ‘1’) {		
texCoord_umin	32	bslbf
texCoord_vmin	32	bslbf
texCoord_size	32	bslbf
}		
texCoord_quant	5	uimsbf
texCoord_pred_type	2	uimsbf
if (texCoord_pred_type==”tree_prediction” || texCoord_pred_type==”parallelogram_prediction”) {		
texCoord_nlambda	2	uimsbf
for (i=1; i<texCoord_nlambda; i++)		
texCoord_lambda	4-19	simsbf
}		
}		
}		

6.2.11.9 ce_SNHC _header

ce_SNHC_header() {	No. of bits	Mnemonic
ce_SNHC_n_proj_surface_spheres	4	uimsbf
if (ce_SNHC_n_proj_surface_spheres != 0) {		
ce_SNHC_x_coord_center_point	32	bslbf
ce_SNHC_y_coord_center_point	32	bslbf
ce_SNHC_z_coord_center_point	32	bslbf
ce_SNHC_normalized_screen_distance_factor	8	uimsbf
for (i=0; i< ce_SNHC_n_proj_surface_spheres; i++) {		
ce_SNHC_radius	32	bslbf
ce_SNHC_min_proj_surface	32	bslbf
ce_SNHC_n_proj_points	8	uimsbf
for (j=0; j< ce_SNHC _n_proj_points; j++) {		
ce_SNHC_sphere_point_coord	11	uimsbf
ce_SNHC_proj_surface	32	bslbf
}		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.11.10 connected_component

connected_component() {	No. of bits	Mnemonic
vertex_graph()		
qf_decode(**has_stitches**, has_stitches_context)		vlclbf
if (has_stitches == '1')		
stitches()		
triangle_tree()		
triangle_data()		
}		

6.2.11.11 vertex_graph

vertex_graph() {	No. of bits	Mnemonic
qf_decode(**vg_simple**, vg_simple_context)		vlclbf
depth = 0		
code_last = '1'		
openloops = 0		
do {		
do {		
if (code_last == '1') {		
qf_decode(**vg_last**, vg_last_context)		vlclbf
if (openloops > 0) {		
qf_decode(**vg_forward_run**, vg_forward_run_context)		vlclbf
if (vg_forward_run == '0') {		
openloops--		
if (openloops > 0)		
qf_decode(**vg_loop_index**, vg_loop_index_context)		vlclbf
break		
}		
}		
}		
qf_decode(**vg_run_length**, vg_run_length_context)		vlclbf
qf_decode(**vg_leaf**, vg_leaf_context)		vlclbf
if (vg_leaf == '1' && vg_simple == '0') {		
qf_decode(**vg_loop**, vg_loop_context)		vlclbf
if (vg_loop == '1')		
openloops++		
}		
} while (0)		
if (vg_leaf == '1' && vg_last == '1' && code_last == '1')		
depth--		
if (vg_leaf == '0' && (vg_last == '0' || code_last == '0'))		
depth++		
code_last = vg_leaf		
} while (depth >= 0)		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.11.12 stitches

stitches() {	No. of bits	Mnemonic
for each vertex in connected_component {		
qf_decode(**stitch_cmd**, stitch_cmd_context)		vlclbf
if (stitch_cmd) {		
qf_decode(**stitch_pop_or_get**, stitch_pop_or_get_context)		vlclbf
if (stitch_pop_or_get == '1') {		
qf_decode(**stitch_pop**, stitch_pop_context)		vlclbf
qf_decode(**stitch_stack_index,** stitch_stack_index_context)		vlclbf
qf_decode(**stitch_incr_length**, stitch_incr_length_context)		vlclbf
if (stitch_incr_length != 0)		
qf_decode(**stitch_incr_length_sign**, stitch_incr_length_sign_context)		vlclbf
qf_decode(**stitch_push**, stitch_push_context)		vlclbf
if (total length >0)		
qf_decode(**stitch_reverse**, stitch_reverse_context)		vlclbf
}		
else		
qf_decode(**stitch_length**, stitch_length_context)		vlclbf
}		
}		
}		

6.2.11.13 triangle_tree

triangle_tree() {	No. of bits	Mnemonic
depth = 0		
ntriangles = 0		
branch_position = -2		
do {		
qf_decode(**tt_run_length**, tt_run_length_context)		vlclbf
ntriangles += tt_run_length		
qf_decode(**tt_leaf**, tt_leaf_context)		vlclbf
if (tt_leaf == '1') {		
depth--		
}		
else {		
branch_position = ntriangles		
depth++		
}		
} while (depth >= 0)		
if (3D_MOBL_start_code == "partition_type_2")		
if (codap_right_bloop_idx – codap_left_bloop_idx – 1 > ntriangles) {		
if (branch_position == ntriangles – 2) {		
qf_decode(**codap_branch_len**, codap_branch_len_context)		vlclbf

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ntriangles -= 2		
}		
else		
ntriangles--		
}		
}		

6.2.11.14 triangle_data

triangle_data(i) {	**No. of bits**	**Mnemonic**
qf_decode(**triangulated**, triangulated_context)		vlclbf
depth=0		
root_triangle()		
for (i=1; i<ntriangles; i++)		
triangle(i)		
}		

6.2.11.15 root_triangle

root_triangle() {	**No. of bits**	**Mnemonic**
if (marching_triangle)		
qf_decode(**marching_pattern**, marching_pattern_context[**marching_pattern**])		vlclbf
else {		
if (3D_MOBL_start_code == “partition_type_2”)		
if (tt_leaf == ‘0’ && depth==0)		
qf_decode(**td_orientation**, td_orientation_context)		vlclbf
if (tt_leaf == ‘0’)		
depth++		
else		
depth--		
}		
if (3D_MOBL_start_code == “partition_type_2”)		
if (triangulated == ‘0’)		
qf_decode(**polygon_edge**, polygon_edge_context[**polygon_edge**])		vlclbf
root_coord()		
root_normal()		
root_color()		
root_texCoord()		
}		

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

root_coord() {	No. of bits	Mnemonic
if (3D_MOBL_start_code == “partition_type_2”) {		
if (visited[vertex_index] == 0) {		
root_coord_sample()		
if (visited[vertex_index] == 0) {		
coord_sample()		
coord_sample()		
}		
}		
}		
else {		
root_coord_sample()		
coord_sample()		
coord_sample()		
}		
}		

root_normal() {	No. of bits	Mnemonic
if (normal_binding != “not_bound”)		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (normal_binding != “bound_per_vertex” || visited[vertex_index] == 0) {		
root_normal_sample()		

if (normal_binding != “bound_per_face” &&
(normal_binding != “bound_per_vertex” ||
visited[vertex_index] == 0)) {		

normal_sample()		
normal_sample()		
}		
}		
else {		
root_normal_sample()		
if (normal_binding != “bound_per_face”) {		
normal_sample()		
normal_sample()		
}		
}		
}		

root_color() {	No. of bits	Mnemonic
if (color_binding != “not_bound”)		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (color_binding != “bound_per_vertex” || visited[vertex_index] == 0) {		
root_color_sample()		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (color_binding != “bound_per_face” &&
(color_binding != “bound_per_vertex” ||
visited[vertex_index] == 0)) {		

color_sample()		
color_sample()		
}		
}		
else {		
root_color_sample()		
if (color_binding != “bound_per_face”) {		
color_sample()		
color_sample()		
}		
}		
}		

root_texCoord() {	**No. of bits**	**Mnemonic**
if (texCoord_binding != “not_bound”)		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (texCoord_binding != “bound_per_vertex” || visited[vertex_index] == 0) {		
root_texCoord_sample()		
if (texCoord_binding!= “bound_per_vertex” || visited[vertex_index] == 0) {		
texCoord_sample()		
texCoord_sample()		
}		
}		
else {		
root_texCoord_sample()		
texCoord_sample()		
texCoord_sample()		
}		
}		

6.2.11.16 triangle

triangle(i) {	**No. of bits**	**Mnemonic**
if (marching_triangle)		
qf_decode(**marching_edge**, marching_edge_context[**marching_edge**])		vlclbf
else {		
if (3D_MOBL_start_code == “partition_type_2”)		
if (tt_leaf == ‘0’ && depth==0)		
qf_decode(**td_orientation**, td_orientation_context)		vlclbf
if (tt_leaf == ‘0’)		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

depth++		
else		
depth--		
if (triangulated == '0')		
qf_decode(**polygon_edge**, polygon_edge_context[**polygon_edge**])		vlclbf
coord()		
normal()		
color()		
texCoord()		
}		

coord() {	**No. of bits**	**Mnemonic**
if (3D_MOBL_start_code == "partition_type_2") {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_coord_sample()		
else		
coord_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
coord_sample()		
}		
}		

normal() {	**No. of bits**	**Mnemonic**
if (normal_binding == "bound_per_vertex") {		
if (3D_MOBL_start_code == "partition_type_2") {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_normal_sample()		
else		
normal_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
normal_sample()		
}		
} else if (normal_binding == "bound_per_face") {		
if (triangulated == '1' || polygon_edge == '1')		
normal_sample()		
} else if (normal_binding == "bound_per_corner") {		
if (triangulated == '1' || polygon_edge == '1') {		
normal_sample()		
normal_sample()		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}		
normal_sample()		
}		
}		

color() {	No. of bits	Mnemonic
if (color_binding == “bound_per_vertex”) {		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_color_sample()		
else		
color_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
color_sample()		
}		
} else if (color_binding == “bound_per_face”) {		
if (triangulated == ‘1’ || polygon_edge == ‘1’)		
color_sample()		
} else if (color_binding == “bound_per_corner”) {		
if (triangulated == ‘1’ || polygon_edge == ‘1’) {		
color_sample()		
color_sample()		
}		
color_sample()		
}		
}		

texCoord() {	No. of bits	Mnemonic
if (texCoord_binding == “bound_per_vertex”) {		
if (3D_MOBL_start_code == “partition_type_2”) {		
if (visited[vertex_index] == 0)		
if (no_ancestors)		
root_texCoord_sample()		
else		
texCoord_sample()		
}		
else {		
if (visited[vertex_index] == 0)		
texCoord_sample()		
}		
} else if (texCoord_binding == “bound_per_corner”) {		
if (triangulated == ‘1’ || polygon_edge == ‘1’) {		
texCoord_sample()		
texCoord_sample()		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}		
texCoord_sample()		
}		
}		

root_coord_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++)		
for (j=0; j<coord_quant; j++)		
qf_decode(**coord_bit**, zero_context)		vlclbf
}		

root_normal_sample() {	No. of bits	Mnemonic
for (i=0; i<1; i++)		
for (j=0; j<normal_quant; j++)		
qf_decode(**normal_bit**, zero_context)		vlclbf
}		

root_color_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++)		
for (j=0; j<color_quant; j++)		
qf_decode(**color_bit**, zero_context)		vlclbf
}		

root_texCoord_sample() {	No. of bits	Mnemonic
for (i=0; i<2; i++)		
for (j=0; j<texCoord_quant; j++)		
qf_decode(**texCoord_bit**, zero_context)		vlclbf
}		

coord_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++) {		
j=0		
do {		
qf_decode(**coord_leading_bit**, coord_leading_bit_context[3*j+i])		vlclbf
j++		
} while (j<coord_quant && coord_leading_bit == ‘0’)		
if (coord_leading_bit == ‘1’) {		
qf_decode(**coord_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**coord_trailing_bit**, zero_context)		vlclbf
} while (j<coord_quant)		
}		
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

normal_sample() {	No. of bits	Mnemonic
for (i=0; i<1; i++) {		
j=0		
do {		
qf_decode(**normal_leading_bit**, normal_leading_bit_context[j])		vlclbf
j++		
} while (j<normal_quant && normal_leading_bit == '0')		
if (normal_leading_bit == '1') {		
qf_decode(**normal_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**normal_trailing_bit**, zero_context)		vlclbf
} while (j<normal_quant)		
}		
}		
}		

color_sample() {	No. of bits	Mnemonic
for (i=0; i<3; i++) {		
j=0		
do {		
qf_decode(**color_leading_bit**, color_leading_bit_context[3*j+i])		vlclbf
j++		
} while (j<color_quant && color_leading_bit == '0')		
if (color_leading_bit == '1') {		
qf_decode(**color_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**color_trailing_bit**, zero_context)		vlclbf
} while (j<color_quant)		
}		
}		
}		

texCoord_sample() {	No. of bits	Mnemonic
for (i=0; i<2; i++) {		
j=0		
do {		
qf_decode(**texCoord_leading_bit**, texCoord_leading_bit_context[2*j+i])		vlclbf
j++		
} while (j<texCoord_quant && texCoord_leading_bit == '0')		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

if (texCoord_leading_bit == '1') {		
qf_decode(**texCoord_sign_bit**, zero_context)		vlclbf
do {		
qf_decode(**texCoord_trailing_bit**, zero_context)		vlclbf
} while (j<texCoord_quant)		
}		
}		
}		

6.2.11.17 3DMeshObject_Refinement_Layer

3DMeshObject_Refinement_Layer () {	**No. of bits**	**Mnemonic**
do {		
3D_MORL_start_code	16	uimsbf
morl_id	8	uimsbf
connectivity_update	2	uimsbf
pre_smoothing	1	bslbf
if (pre_smoothing == '1')		
pre_smoothing_parameters()		
post_smoothing	1	bslbf
if (post_smoothing == '1')		
post_smoothing_parameters()		
while (!bytealigned())		
one_bit	1	bslbf
qf_start()		
if (connectivity_update == "fs_update")		
fs_pre_update()		
if (pre_smoothing == '1' || post_smoothing == '1')		
smoothing_constraints()		
/* apply pre smoothing step */		
if (connectivity_update == "fs_update")		
fs_post_update()		
if (connectivity_update != "not_updated")		
qf_decode(**other_update**, zero_context)		vlclbf
if (connectivity_update=="not_updated" || other_update == '1')		
other_property_update()		
/* apply post smoothing step */		
} while (nextbits_bytealigned() == 3D_MORL_start_code)		
}		

6.2.11.17.1 pre_smoothing_parameters

pre_smoothing_parameters() {	**No. of bits**	**Mnemonic**
pre_smoothing_n	8	uimsbf
pre_smoothing_lambda	32	bslbf
pre_smoothing_mu	32	bslbf
}		

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.11.17.2 post_smoothing_parameters

post_smoothing_parameters() {	No. of bits	Mnemonic
post_smoothing_n	8	uimsbf
post_smoothing_lambda	32	bslbf
post_smoothing_mu	32	bslbf
}		

6.2.11.17.3 fs_pre_update

fs_pre_update() {	No. of bits	Mnemonic
for each connected component {		
forest()		
for each tree in forest {		
triangle_tree()		
/* for each tree loop vertex set visited[vertex_index]='1' */		
triangle_data()		
}		
}		
}		

forest () {	No. of bits	Mnemonic
for each edge in connected component		
if (creates no loop in forest)		
qf_decode(**pfs_forest_edge**, pfs_forest_edge_context)		vlclbf
}		

6.2.11.17.4 smoothing_constraints

smoothing_constraints() {	No. of bits	Mnemonic
qf_decode(**smooth_with_sharp_edges**, zero_context)		vlclbf
if (smooth_with_sharp_edges == '1')		
sharp_edge_marks()		
qf_decode(**smooth_with_fixed_vertices**, zero_context)		vlclbf
if (smooth_with_fixed_vertices == '1')		
fixed_vertex_marks()		
}		

sharp_edge_marks () {	No. of bits	Mnemonic
for each edge		
qf_decode(**smooth_sharp_edge**, smooth_sharp_edge_context)		vlclbf
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

fixed_vertex_marks () {	No. of bits	Mnemonic
for each vertex		
qf_decode(**smooth_fixed_vertex**, smooth_fixed_vertex_context)		vlclbf
}		

6.2.11.17.5 fs_post_update

fs_post_update() {	No. of bits	Mnemonic
for each connected component {		
for each tree in forest		
tree_loop_property_update()		
}		

tree_loop_property_update () {	No. of bits	Mnemonic
loop_coord_update()		
loop_normal_update()		
loop_color_update()		
loop_texCoord_update()		
}		

loop_coord_update () {	No. of bits	Mnemonic
for each tree loop vertex		
coord_sample()		
}		

loop_normal_update () {	No. of bits	Mnemonic
if (normal_binding == “bound_per_vertex”) {		
for each tree loop vertex		
normal_sample()		
}		
else if (normal_binding == “bound_per_face”) {		
for each tree loop face		
normal_sample()		
}		
else if (normal_binding == “bound_per_corner”) {		
for each tree loop corner		
normal_sample()		
}		
}		

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

loop_color_update () {	No. of bits	Mnemonic
if (color_binding == “bound_per_vertex”) {		
for each tree loop vertex		
color_sample()		
}		
else if (color_binding == “bound_per_face”) {		
for each tree loop face		
color_sample()		
}		
else if (color_binding == “bound_per_corner”) {		
for each tree loop corner		
color_sample()		
}		
}		

loop_texCoord_update () {	No. of bits	Mnemonic
if (texCoord_binding == “bound_per_vertex”) {		
for each tree loop vertex		
texCoord_sample()		
}		
else if (texCoord_binding == “bound_per_corner”) {		
for each tree loop corner		
texCoord_sample()		
}		
}		

6.2.11.17.6 other_property_update

other_property_update() {	No. of bits	Mnemonic
other_coord_update()		
other_normal_update()		
other_color_update()		
other_texCoord_update()		
}		

other_coord_update () {	No. of bits	Mnemonic
for each vertex in mesh		
if (vertex is not a tree loop vertex)		
coord_sample()		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

other_normal_update () {	No. of bits	Mnemonic
if (normal_binding == “bound_per_vertex”) {		
for each vertex in mesh		
if (vertex is not a tree loop vertex)		
normal_sample()		
}		
else if (normal_binding == “bound_per_face”) {		
for each face in mesh		
if (face is not a tree loop face)		
normal_sample()		
}		
else if (normal_binding == “bound_per_corner”) {		
for each corner in mesh		
if (corner is not a tree loop corner)		
normal_sample()		
}		
}		

other_color_update () {	No. of bits	Mnemonic
if (color_binding == “bound_per_vertex”) {		
for each vertex in mesh		
if (vertex is not a tree loop vertex)		
color_sample()		
}		
else if (color_binding == “bound_per_face”) {		
for each face in mesh		
if (face is not a tree loop face)		
color_sample()		
}		
else if (color_binding == “bound_per_corner”) {		
for each corner in mesh		
if (corner is not a tree loop corner)		
color_sample()		
}		
}		

other_texCoord_update () {	No. of bits	Mnemonic
if (texCoord_binding == “bound_per_vertex”) {		
for each vertex in tree loop		
if (vertex is not a tree loop vertex)		
texCoord_sample()		
}		
else if (texCoord_binding == “bound_per_corner”) {		
for each corner in mesh		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (corner is not a tree loop corner)		
texCoord_sample()		
}		
}		

6.2.12 Upstream message

6.2.12.1 upstream_message

upstream_message() {	No. of bits	Mnemonic
upstream_message_type	3	bslbf
If (upstream_message_type == Video_NEWPRED)		
upstream_Video_NEWPRED()		
if (upstream_message_type == SNHC_QoS)		
upstream_SNHC_QoS()		
byte_align_for_upstream()		
}		

6.2.12.2 upstream_Video_NEWPRED

upstream_Video_NEWPRED() {	No. of bits	Mnemonic
newpred_upstream_message_type	2	bslbf
if (newpred_upstream_message_type == “NP_NACK”)		
unreliable_flag	1	bslbf
vop_id	4-15	uimsbf
macroblock_number	1-14	uimsbf
if (newpred_upstream_message_type == “Intra Refresh Command”)		
end_macroblock_number	1-14	uimsbf
if (newpred_upstream_message_type == “NP_NACK”)		
requested_vop_id_for_prediction	4-15	uimsbf
}		

6.2.12.3 upstream_SNHC_QoS

upstream_SNHC_QoS() {	No. of bits	Mnemonic
screen_width	12	uimsbf
screen_height	12	uimsbf
n_rendering_modes	4	uimsbf
for (rendering_mode = 0 ; rendering_mode < n_rendering_modes ; rendering_mode++) {		
rendering_mode_type	4	uimsbf
n_curves	8	uimsbf
for (i = 0 ; i < n_curves ; i++) {		
triangle_parameter	24	uimsbf
n_points_on_curve	8	uimsbf

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

for (j = 0 ; j < n_points_on_curve ; j++) {		
screen_coverage_parameter	8	uimsbf
frame_rate_value	12	bslbf
}		
}		
}		
}		

6.2.13 Studio Video Object

6.2.13.1 Studio Visual Object

StudioVisualObject() {	**No. of bits**	**Mnemonic**
visual_object_start_code	32	bslbf
visual_object_verid	4	uimsbf
visual_object_type	4	uimsbf
next_start_code_studio()		
extension_and_user_data(1)		
if (visual_object_type == “video ID“) {		
video_object_start_code	32	bslbf
StudioVideoObjectLayer()		
} else {		
/* Other visual object types are not supported in StudioVisualObject() */		
}		
}		

6.2.13.2 Extension and user data

extension_and_user_data(i) {	**No. of bits**	**Mnemonic**
while ((next_bits() == extension_start_code) ||		
(next_bits() == user_data_start_code)) {		
if ((i==2 || i==4) && (next_bits() == extension_start_code))		
extension_data(i)		
if (next_bits() == user_data_start_code)		
user_data_studio()		
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.13.2.1 Extension data

extension_data(i) {	No. of bits	Mnemonic
while (next_bits()== extension_start_code) {		
extension_start_code	32	bslbf
/* NOTE - i never takes the value 0 because extension_data() is never called in a VisualObjectSequence() */		
/* NOTE - i never takes the value 1 because extension_data() is never called in a StudioVisualObject() */		
if (i == 2) { /* Called in StudioVideoObjectLayer() */		
if (next_bits()== “Sequence Display Extension ID”)		
sequence_display_extension()		
else if (next_bits() == “Quant Matrix Extension ID”)		
quant_matrix_extension()		
else if (nextbits() == “VLC Code Extension ID”)		
vlc_code_extension()		
}		
/* NOTE - i never takes the value 3 because extension_data() is never called in a Group_of_StudioVideoObjectPlane() */		
if (i == 4) { /* Called in VideoObjectPlane() */		
if (nextbits() == “Quant Matrix Extension ID”)		
quant_matrix_extension()		
else if (nextbits() == “Copyright Extension ID”)		
copyright_extension()		
else if (nextbits() == “Picture Display Extension ID”)		
picture_display_extension()		
else if(nextbits() == “Camera Prameters Extension ID”)		
camera_parameters_extension()		
else if (nextbits() == “ITU-T Extension ID”)		
ITU-T_extension()		
else if (nextbits() == “VLC Code Extension ID”)		
vlc_code_extension()		
}		
}		

6.2.13.2.2 User data Studio

user_data_studio() {	No. of bits	Mnemonic
user_data_start_code	32	bslbf
while(next_bits() != ‘0000 0000 0000 0000 0000 0001’) {		
user_data	8	uimsbf
}		
next_start_code_studio()		
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.13.2.3 Sequence display extension

sequence_display_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
video_format	3	uimsbf
video_range	1	bslbf
colour_description	1	uimsbf
if (colour_description) {		
colour_primaries	8	uimsbf
transfer_characteristics	8	uimsbf
matrix_coefficients	8	uimsbf
}		
display_horizontal_size	14	uimsbf
marker_bit	1	bslbf
display_vertical_size	14	uimsbf
next_start_code_studio()		
}		

6.2.13.2.4 Quant matrix extension

quant_matrix_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
load_intra_quantiser_matrix	1	uimsbf
if (load_intra_quantiser_matrix)		
intra_quantiser_matrix[64]	8 * 64	uimsbf
load_non_intra_quantiser_matrix	1	uimsbf
if (load_non_intra_quantiser_matrix)		
non_intra_quantiser_matrix[64]	8 * 64	uimsbf
load_chroma_intra_quantiser_matrix	1	uimsbf
if (load_chroma_intra_quantiser_matrix)		
chroma_intra_quantiser_matrix[64]	8 * 64	uimsbf
load_chroma_non_intra_quantiser_matrix	1	uimsbf
if (load_chroma_non_intra_quantiser_matrix)		
chroma_non_intra_quantiser_matrix[64]	8 * 64	uimsbf
if (video_object_layer_shape == 'grayscale') {		
for(i=0; i<aux_comp_count; i++) {		
load_intra_quantiser_matrix_grayscale[i]	1	uimsbf
if (load_intra_quantiser_matrix_grayscale[i])		
intra_quantiser_matrix_grayscale [i][64]	8 * 64	uimsbf
load_non_intra_quantiser_matrix_grayscale[i]	1	uimsbf
if (load_non_intra_quantiser_matrix_grayscale[i])		
non_intra_quantiser_matrix_grayscale [i][64]	8 * 64	uimsbf
}		
}		
next_start_code_studio()		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.13.2.5 Picture display extension

picture_display_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
for (i=0; i<number_of_frame_centre_offsets; i++) {		
frame_centre_horizontal_offset	16	simsbf
marker_bit	1	bslbf
frame_centre_vertical_offset	16	simsbf
marker_bit	1	bslbf
}		
next_start_code_studio()		
}		

6.2.13.2.6 Copyright extension

copyright_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
copyright_flag	1	uimsbf
copyright_identifier	8	uimsbf
original_or_copy	1	uimsbf
reserved	7	bslbf
marker_bit	1	bslbf
copyright_number_1	20	uimsbf
marker_bit	1	bslbf
copyright_number_2	22	uimsbf
marker_bit	1	bslbf
copyright_number_3	22	uimsbf
next_start_code_studio()		
}		

6.2.13.2.7 Camera Parameters extension

camera_parameters_extension() {	No. of bits	Mnemonic
extension_start_code_identifier	4	uimsbf
reserved	1	uimsbf
camera_id	7	simsbf
marker_bit	1	bslbf
height_of_image_device	22	uimsbf
marker_bit	1	bslbf
focal_length	22	uimsbf
marker_bit	1	bslbf
f_number	22	uimsbf
marker_bit	1	bslbf
vertical_angle_of_view	22	uimsbf
marker_bit	1	bslbf
camera_position_x_upper	16	simsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

camera_position_x_lower	16	simsbf
marker_bit	1	bslbf
camera_position_y_upper	16	simsbf
marker_bit	1	bslbf
camera_position_y_lower	16	simsbf
marker_bit	1	bslbf
camera_position_z_upper	16	simsbf
marker_bit	1	bslbf
camera_position_z_lower	16	simsbf
marker_bit	1	bslbf
camera_direction_x	22	simsbf
marker_bit	1	bslbf
camera_direction_y	22	simsbf
marker_bit	1	bslbf
camera_direction_z	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_x	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_y	22	simsbf
marker_bit	1	bslbf
image_plane_vertical_z	22	simsbf
marker_bit	1	bslbf
reserved	32	bslbf
next_start_code_studio()		
}		

6.2.13.2.8 ITU-T extension

ITU-T extension () {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
while (nextbits() != '0000 0000 0000 0000 0000 0001') {		
ITU-T_data	1	bslbf
}		
next_start_code_studio()		
}		
NOTE – The construct with the while-statement prevents start code emulation.		

6.2.13.2.9 VLC code extension

vlc_code_extension() {	**No. of bits**	**Mnemonic**
extension_start_code_identifier	4	uimsbf
load_vlc_code	4	uimsbf
if (load_intra_vlc_code == 1) {		
for (j=0; j< 12; j++) {		
for (i=0; i< 22; i++) {		
Intra_vlc_length[j][i]	4	uimsbf

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Intra_vlc_code[j][i]	16	uimsbf
}		
}		
}		
if (load_inter_vlc_code == 1) {		
for (j=0; j< 12; j++) {		
for (i=0; i< 22; i++) {		
inter_vlc_length[j][i]	4	uimsbf
inter_vlc_code[j][i]	16	uimsbf
}		
}		
}		
next_start_code_studio()		
}		

6.2.13.3 Studio Video Object Layer

StudioVideoObjectLayer() {	**No. of bits**	**Mnemonic**
video_object_layer_start_code	32	bslbf
random_accessible_vol	1	bslbf
video_object_type_indication	8	uimsbf
video_object_layer_verid	4	uimsbf
video_object_layer_shape	2	uimsbf
video_object_layer_shape_extension	4	uimsbf
progressive_sequence	1	bslbf
if (video_object_layer_shape != “binary only”) {		
rgb_components	1	uimsbf
chroma_format	2	uimsbf
bits_per_pixel	4	uimsbf
}		
if (video_object_layer_shape == “rectangular”) {		
marker_bit	1	bslbf
video_object_layer_width	14	uimsbf
marker_bit	1	bslbf
video_object_layer_height	14	uimsbf
marker_bit	1	bslbf
}		
aspect_ratio_info	4	uimsbf
if (aspect_ratio_info == “extended_PAR”) {		
par_width	8	uimsbf
par_height	8	uimsbf
}		
frame_rate_code	4	uimsbf
first_half_bit_rate	15	uimsbf
marker_bit	1	bslbf
latter_half_bit_rate	15	uimsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

first_half_vbv_buffer_size	15	uimsbf
marker_bit	1	bslbf
latter_half_vbv_buffer_size	3	uimsbf
first_half_vbv_occupancy	11	uimsbf
marker_bit	1	blsbf
latter_half_vbv_occupancy	15	uimsbf
marker_bit	1	blsbf
low_delay	1	uimsbf
mpeg2_stream	1	uimsbf
if (video_object_layer_shape == ”grayscale”) {		
for(i=0; i<aux_comp_count; i++) {		
alpha_bits_per_pixel[i]	4	uimsbf
minimum_alpha_level[i]	12	uimsbf
marker_bit	1	blsbf
maximum_alpha_level[i]	12	uimsbf
}		
composition_method	1	bslbf
linear_composition	1	bslbf
}		
if (video_object_layer_shape != “binary only”) {		
sprite_enable	1	bslbf
if (sprite_enable) {		
sprite_width	20	uimsbf
marker_bit	1	bslbf
sprite_height	20	uimsbf
marker_bit	1	bslbf
sprite_left_coordinate	20	uimsbf
marker_bit	1	bslbf
sprite_top_coordinate	20	uimsbf
marker_bit	1	bslbf
no_of_sprite_warping_points	6	uimsbf
sprite_warping_accuracy	2	uimsbf
sprite_brightness_change	1	bslbf
sprite_defocusing	1	bslbf
sprite_lens_distortion	1	bslbf
}		
}		
next_start_code_studio()		
extension_and_user_data(2)		
if (sprite_enable)		
StudioVideoObjectPlane()		
do {		
if (next_bits() == group_of_vop_start_code)		
Group_of_StudioVideoObjectPlane()		
StudioVideoObjectPlane()		
} while ((next_bits()== group_of_vop_start_code) || (next_bits() == vop_start_code))		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.13.4 Group of Studio Video Object Plane

Group_of_StudioVideoObjectPlane() {	No. of bits	Mnemonic
group_vop_start_codes	32	bslbf
time_code_smpte12m()		
closed_gov	1	bslbf
broken_link	1	bslbf
next_start_code_studio()		
extension_and_user_data(3)		
}		

6.2.13.4.1 Time Code SMPTE12M

time_code_smpte12m(){	Bits	Mnemonic
Time_code[63..48]	16	uimsbf
marker_bit	1	bslbf
Time_code[47..32]	16	uimsbf
marker_bit	1	bslbf
Time_code[31..16]	16	uimsbf
marker_bit	1	bslbf
Time_code[15..0]	16	uimsbf
marker_bit	1	bslbf
reserved_bits	4	bslbf
}		

6.2.13.5 Studio Video Object Plane

StudioVideoObjectPlane() {	No. of bits	Mnemonic
vop_start_code	32	bslbf
time_code_smpte12m()		
temporal_reference	10	bslbf
vop_structure	2	uimsbf
vop_coding_type	2	uimsbf
vop_coded	1	bslbf
if (vop_coded == '0') {		
next_start_code_studio()		
extension_and_user_data(4)		
return()		
}		
if (video_object_layer_shape != "rectangular") {		
if (!(sprite_enable && vop_coding_type == "I")) {		
vop_width	14	uimsbf
marker_bit	1	bslbf
vop_height	14	uimsbf
marker_bit	1	bslbf
vop_horizontal_mc_spatial_ref	14	simsbf
marker_bit	1	bslbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

vop_vertical_mc_spatial_ref	14	simsbf
marker_bit	1	bslbf
}		
}		
top_field_first	1	bslbf
repeat_first_field	1	bslbf
progressive_frame	1	bslbf
if (vop_coding_type == "I")		
intra_predictors_reset	1	bslbf
}		
if (video_object_layer_shape != “binary only”) {		
alternate_scan	1	bslbf
frame_pred_frame_dct	1	bslbf
dct_precision	2	uimsbf
intra_dc_precision	2	uimsbf
q_scale_type	1	bslbf
if ((vop_coding_type != "I")		
&& !(sprite_enable && vop_coding_type == "S")) {		
vop_fcode[0][0]	4	uimsbf
vop_fcode[0][1]	4	uimsbf
vop_fcode[1][0]	4	uimsbf
vop_fcode[1][1]	4	uimsbf
dead_zone_disable	1	bslbf
}		
}		
if (video_object_layer_shape == “grayscale”) {		
marker_bit	1	bslbf
for(i=0; i<aux_comp_count; i++) {		
alpha_dct_precision[i]	2	uimsbf
alpha_intra_dc_precision[i]	2	uimsbf
alpha_q_scale_type[i]	1	bslbf
}		
}		
if (sprite_enable && vop_coding_type == “S”) {		
if (no_sprite_points > 0)		
sprite_trajectory()		
if (sprite_brightness_change) {		
vop_sprite_brightness_change	1	bslbf
if (vop_sprite_brightness_change)		
brightness_change_factor()		
}		
if (sprite_defocusing)		
defocusing_control()		
if (sprite_lens_distortion)		
lens_distortion_parameter()		
next_start_code_studio()		
return()		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

composite_display_flag	1	bslbf
if (composite_display_flag) {		
v_axis	1	bslbf
field_sequence	3	uimsbf
sub_carrier	1	bslbf
burst_amplitude	7	uimsbf
sub_carrier_phase	8	uimsbf
}		
while (nextbits() == '1') {		
extra_bit_picture /* with the value '1' */	1	uimsbf
extra_information_picture	8	uimsbf
}		
extra_bit_picture /* with the value '0' */	1	uimsbf
next_start_code_studio()		
extension_and_user_data(4)		
do {		
StudioSlice()		
} while (nextbits() == slice_start_code)		
}		

6.2.13.6 Studio sprite coding

sprite_trajectory() {	**No. of bits**	**Mnemonic**
for (i=0; i < no_of_sprite_warping_points; i++) {		
warping_mv_code(du[i])		
warping_mv_code(dv[i])		
}		
}		

warping_mv_code(d) {	**No. of bits**	**Mnemonic**
dmv_length	2-19	uimsbf
if (dmv_length != '00')		
dmv_code	1-21	uimsbf
marker_bit	1	bslbf
}		

brightness_change_factor() {	**No. of bits**	**Mnemonic**
brightness_change_factor_size	1-4	uimsbf
brightness_change_factor_code	5-10	uimsbf
}		

defocusing_control() {	**No. of bits**	**Mnemonic**
marker_bit	1	bslbf
defocusing_control_parameter	9-12	uimsbf
}		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

lens_distortion_parameter() {	No. of bits	Mnemonic
marker_bit	1	bslbf
lens_distortion_parameter_1	16	bslbf
marker_bit	1	bslbf
lens_distortion_parameter_2	16	bslbf
marker_bit	1	bslbf
lens_center_horizontal	14	bslbf
marker_bit	1	bslbf
lens_center_vertical	14	bslbf
}		

6.2.13.7 Studio Slice

StudioSlice() {	No. of bits	Mnemonic
slice_start_code	32	uimsbf
macroblock_number	1-14	vlclbf
if (video_object_layer_shape != “binary only”)		
quantiser_scale_code	5	uimsbf
if(video_object_layer_shape==”grayscale”)		
for(i=0; i<aux_comp_count; i++)		
alpha_quantiser_scale_code[i]	5	uimsbf
if (nextbits() == 1) {		
slice_extension_flag	1	bslbf
intra_slice	1	uimsbf
slice_VOP_id_enable	1	uimsbf
slice_VOP_id	6	uimsbf
while (nextbits() == 1) {		
extra_bit_slice	1	uimsbf
extra_information_slice	8	uimsbf
}		
}		
extra_bit_slice	1	uimsbf
do {		
StudioMacroblock()		
} while (next_bits() != ‘000 0000 0000 0000 0000 0000’)		
next_start_code_studio()		
}		

6.2.13.8 Studio Macroblock

StudioMacroblock() {	No. of bits	Mnemonic
if (vop_coding_type != “B”) {		
if (video_object_layer_shape != “rectangular”)		
Studio_mb_binary_shape_coding ()		
if (video_object_layer_shape != “binary only”) {		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (!transparent_mb()) {		
if (vop_coding_type != “I”)		
not_coded	1	bslbf
if (vop_coding_type == “I” || !not_coded) {		
compression_mode	1	bslbf
if (compression_mode == "DCT") { /* DCT */		
StudioMacroblock_modes()		
if (macroblock_quant)		
quantiser_scale_code	5	uimsbf
if (macroblock_motion_forward)		
motion_vectors(0)		
if (macroblock_pattern)		
coded_block_pattern()		
for (i = 0; i < block_count; i++)		
if(!transparent_block(i))		
StudioBlock(i)		
} /* -end- DCT */		
else { /* DPCM */		
marker_bit	1	bslbf
dpcm_scan_order	1	bslbf
StudioDPCMBlock() // Y or G		
StudioDPCMBlock() // U or B		
StudioDPCMBlock() // V or R		
} /* -end- DPCM */		
}		
}		
}		
}		
if(video_object_layer_shape==“grayscale” && !transparent_mb()) {		
for (j=0 ; j<aux_comp_count ; j++) {		
if(macroblock_intra || compression_mode == "DPCM") {		
coda_i[j]	1	bslbf
if(coda_i[j]==”coded”) {		
alpha_compression_mode[j]	1	bslbf
if(alpha_compression_mode[j]== "DCT") {		
alpha_macroblock_quant[j]	1	bslbf
if (alpha_macroblock_quant[j])		
alpha_quantiser_scale_code[j]	5	uimsbf
for(i=0;i<alpha_block_count;i++)		
if(!transparent_block(i))		
StudioAlphaBlock(j,i)		
} /* -end- DCT */		
else { /* DPCM */		
marker_bit	1	bslbf
alpha_dpcm_scan_order[j]	1	bslbf
StudioDPCMBlock() // alpha block		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

} /* -end- DPCM */		
}		
} else { /* inter macroblock */		
coda_pb[j]	1-2	vlclbf
if(coda_pb[j]==”coded”){		
alpha_compression_mode[j]	1	bslbf
if(alpha_compression_mode[j]== "DCT") {		
cbpa[j]	1-6	vlclbf
if (cbpa[j] != ‘no DCT coeff’) {		
alpha_macroblock_quant[j]	1	bslbf
if (alpha_macroblock_quant[j])		
alpha_quantiser_scale_code[j]	5	uimsbf
}		
for(i=0 ;i<alpha_block_count ;i++)		
if(!transparent_block(i))		
StudioAlphaBlock(j,i)		
} else {		
marker_bit	1	bslbf
alpha_dpcm_scan_order[j]	1	bslbf
StudioDPCMBlock(j) // Alpha		
}		
}		
} /* -end- inter */		
} /* -roop- aux_comp_count */		
} /* -end- grayscale */		
}		
NOTE The value of block_count is 6 in the 4:2:0 format. The value of alpha_block_count is 4.		

6.2.13.8.1 Studio MB Binary Shape Coding

Studio_mb_binary_shape_coding () {	**No. of bits**	**Mnemonic**
bab_type	1-6	vlclbf
if (vop_coding_type == 'P'){		
if (bab_type == 1){		
mvs_x	1-18	vlclbf
mvs_y	1-18	vlclbf
}		
}		
if (bab_type == 4) {		
inferior_symbol_macroblock	1	bslbf
cbbp	3-7	vlclbf
for (i=0: i<4; i++){		
if (coded_binary_block_pattern[i]==’1’){		
inferior_symbol_block	1	bslbf
scan_direction	1	bslbf
backward_load_flag	1	bslbf
clp	3-24	vlclbf

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

marker_bit	1	blsbf
for (j=0; j<8; j++){		
if (coded_line_pattern[j]=='1') {		
lbp	1-23	vlclbf
marker_bit	1	blsbf
}		
}		
}		
}		
}		
}		

6.2.13.8.2 Studio Macroblock modes

StudioMacroblock_modes() {	**No. of bits**	**Mnemonic**
macroblock_type	1-9	vlclbf
if (macroblock_motion_forward) {		
if (vop_structure == 'frame') {		
if (frame_pred_frame_dct == 0)		
frame_motion_type	2	uimsbf
} else {		
field_motion_type	2	uimsbf
}		
}		
if ((vop_structure == "Frame picture") &&		
(frame_pred_frame_dct == 0) &&		
(macroblock_intra || macoblock_pattern)){		
dct_type	1	uimsbf
}		
}		

6.2.13.8.3 Motion vectors

motion_vectors (s) {	**No. of bits**	**Mnemonic**
if (motion_vector_count == 1) {		
if (mv_format == field)		
motion_vertical_field_select[0][s]	1	uimsbf
motion_vector(0, s)		
} else {		
motion_vertical_field_select[0][s]	1	uimsbf
motion_vector(0, s)		
motion_vertical_field_select[1][s]	1	uimsbf
motion_vector(1, s)		
}		
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

6.2.13.8.4 Motion vector

motion_vector (r, s) {	No. of bits	Mnemonic
motion_code[r][s][0]	1-11	vlclbf
if ((vop_fcode[s][0] != 1) && (motion_code[r][s][0] != 0))		
motion_residual[r][s][0]	1-8	uimsbf
motion_code[r][s][1]	1-11	vlclbf
if ((vop_fcode[s][1] != 1) && (motion_code[r][s][1] != 0))		
motion_residual[r][s][1]	1-8	uimsbf
}		

6.2.13.8.5 Coded block pattern

coded_block_pattern () {	No. of bits	Mnemonic
coded_block_pattern_420	3-9	vlclbf
if (chroma_format == 4:2:2)		
coded_block_pattern_1	2	uimsbf
if (chroma_format == 4:4:4)		
coded_block_pattern_2	6	uimsbf
}		

6.2.13.9 Studio Block

The detailed syntax for the term “DCT coefficient” is fully described in Clause 7.

StudioBlock(i) {	No. of bits	Mnemonic
if (pattern_code[i]) {		
if (macroblock_intra) {		
if (i<4) {		
dct_dc_size_luminance	2-13	vlclbf
if(dct_dc_size_luminance != 0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_luminance > 8)		
marker_bit	1	bslbf
} else {		
dct_dc_size_chrominance	2-13	vlclbf
if(dct_dc_size_chrominance !=0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_chrominance > 8)		
marker_bit	1	bslbf
}		
} else {		
First DCT coefficient	1-32	vlcfbf
}		
while (next_bits() != End of block)		
Subsequent DCT coefficients	1-32	vlclbf
End of block	1-16	vlcfbf
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.13.9.1 Studio Alpha Block

The syntax for DCT coefficient decoding is the same as for block(i) in 6.2.13.9.

StudioAlphaBlock(j, i) {	No. of bits	Mnemonic
last = 0		
if (alpha_pattern_code[j][i]) {		
if (macroblock_intra || compression_mode == "DPCM") {		
dct_dc_size_alpha	2-13	vlclbf
if(dct_dc_size_alpha != 0)		
dct_dc_differential	1-18	vlclbf
if (dct_dc_size_alpha > 8)		
marker_bit	1	bslbf
} else {		
First DCT coefficient	1-32	vlcfbf
}		
while (next_bits() != End of block)		
Subsequent DCT coefficients	1-32	vlclbf
End of block	1-16	vlcfbf
}		
}		

6.2.13.10 Studio DPCM Block

StudioDPCMBlock() {	No. of bits	Mnemonic
block_mean	8-12	uimsbf
rice_parameter	4	uimsbf
for(i=0 ;i<8 or 16 ; i++) {		
for(j=0 ;j<8 or 16;j++) {		
rice_prefix_code	1-12	vlclbf
if (**rice_prefix_code** == '0000 0000 0001')		
dpcm_residual	4-12	uimsbf
else		
rice_suffix_code	0-11	uimsbf
}		
}		
}		

6.2.14 FGS Video Object

6.2.14.1 FGS Video Object Plane

FGSVideoObjectPlane() {	No. of bits	Mnemonic
fgs_vop_start_code	32	bslbf
fgs_vop_coding_type	2	uimsbf
do {		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

modulo_time_base	1	bslbf
} while (modulo_time_base != '0')		
marker_bit	1	bslbf
vop_time_increment	1-16	uimsbf
marker_bit	1	bslbf
fgs_vop_max_level_y	5	uimsbf
fgs_vop_max_level_u	5	uimsbf
fgs_vop_max_level_v	5	uimsbf
marker_bit	1	bslbf
fgs_vop_number_of_vop_bp_coded	5	uimsbf
fgs_vop_mc_bit_plane_used	5	uimsbf
fgs_vop_selective_enhancement_enable	1	bslbf
if (fgs_vop_coding_type != "I") {		
if (interlaced)		
top_field_first	1	bslbf
vop_fcode_forward	3	uimsbf
if (fgs_vop_coding_type == "B")		
vop_fcode_backward	3	uimsbf
fgs_ref_select_code	2	uimsbf
do {		
fgs_motion_macroblock()		
} while(nextbits_bytealigned() != '000 0000 0000 0000 0000 0000')		
}		
next_start_code()		
if (nextbits_bytealigned () == fgs_bp_start_code) {		
while(nextbits_bytealigned() != '000 0000 0000 0000 0000 0000' || nextbits_bytealigned () == fgs_bp_start_code) {		
if (start_of_bit_plane())		
fgs_bp_start_code	32	bslbf
else {		
if (! fgs_resync_marker_disable && nextbits_bytealigned () == fgs_resync_marker) {		
next_resync_marker()		
fgs_resync_marker	23	uimsbf
fgs_ macroblock_number	1-14	vlclbf
}		
}		
fgs_macroblock()		
}		
next_start_code()		
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.14.2 FGS Motion Macroblock

fgs_motion_macroblock() {	No. of bits	Mnemonic
if (fgs_vop_coding_type == “P”) {		
fgs_not_coded	1	bslbf
if (!fgs_not_coded) {		
fgs_p_mb_type	1	bslbf
if (interlaced)		
fgs_motion_interlaced_information()		
if (fgs_p_mb_type == 0) {		
fgs_motion_vector(“forward”)		
if (interlaced && field_prediction)		
fgs_motion_vector(“forward”)		
}		
if (fgs_p_mb_type == 1) {		
for (j=0; j < 4; j++)		
fgs_motion_vector(“forward”)		
}		
}		
} else {		
fgs_modb	1	bslbf
if (!fgs_modb) {		
fgs_b_mb_type	1-4	vlclbf
if (interlaced)		
fgs_motion_interlaced_information()		
if (fgs_b_mb_type==‘0001’ || fgs_b_mb_type==‘01’) {		
fgs_motion_vector(“forward”)		
if (interlaced && field_prediction)		
fgs_motion_vector(“forward”)		
}		
if (fgs_b_mb_type == ‘01’ || fgs_b_mb_type == ‘001’) {		
fgs_motion_vector(“backward”)		
if (interlaced && field_prediction)		
fgs_motion_vector(“backward”)		
}		
if (fgs_b_mb_type == ‘1’)		
fgs_motion_vector(“direct”)		
}		
}		
}		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

6.2.14.3 FGS Motion Interlaced Information

fgs_motion_interlaced_information( ) {	No. of bits	Mnemonic
fgs_field_prediction	1	bslbf
if (fgs_field_prediction) {		
if (fgs_vop_coding_type == “P” ‖ (fgs_vop_coding_type==“B” && fgs_b_mb_type!=“001”)){		
fgs_forward_top_field_reference	1	bslbf
fgs_forward_bottom_field_reference	1	bslbf
}		
if ((fgs_vop_coding_type == “B”) && (fgs_b_mb_type != “1”)) {		
fgs_backward_top_field_reference	1	bslbf
fgs_backward_bottom_field_reference	1	bslbf
}		
}		
}		

6.2.14.4 FGS Motion Vector

fgs_motion_vector (mode) {	No. of bits	Mnemonic
if (mode == „direct“) {		
horizontal_mv_data	1-13	vlclbf
vertical_mv_data	1-13	vlclbf
}		
else if (mode == „forward“) {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1)&&(horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_forward != 1)&&(vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
else if (mode == „backward“) {		
horizontal_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1)&&(horizontal_mv_data != 0))		
horizontal_mv_residual	1-6	uimsbf
vertical_mv_data	1-13	vlclbf
if ((vop_fcode_backward != 1)&&(vertical_mv_data != 0))		
vertical_mv_residual	1-6	uimsbf
}		
}		

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.14.5 FGS Macroblock

fgs_macroblock() {	No. of bits	Mnemonic
if (fgs_vop_bp_id < 2)		
fgs_cbp	1-9	vlclbf
if (fgs_vop_selective_enhancement_enable==1) {		
if (!mb_shift_factor_received && none_zero_macroblock)		
fgs_shifted_bits	1-5	vlclbf
}		
if (interlaced==1) {		
if (!dct_type_received && non_zero_macroblock)		
fgs_dct_type	1	bslbf
}		
for (i=0; i<6; i++) {		
if (start_decode==1)		
fgs_block()		
}		
}		

NOTE 1 In the syntax of fgs vop, there are three elements: fgs_vop_max_level_y, fgs_vop_max_level_u, and fgs_vop_max_level_v. The maximum value of these three elements is the number of vop-bps in an fgs vop. If any one of the three elements has a smaller value than the number of vop-bps in the fgs vop, the color component corresponding to the smaller value element is absent in one or more vop-bps. If the difference between the number of vop-bps and the value of the element is k, the corresponding color component is absent in the first k vop-bps. There is no need to decode fgs blocks of the color component absent in the vop-bps. start_decode is defined to be a flag to indicate whether decoding should be performed for an fgs block depending on whether the fgs block belongs to an absent color component or not. The value of start_decode is 0 if the fgs block belongs to an absent color component. Otherwise, the value of start_decode is 1 and decoding is performed for the fgs block.

NOTE 2 mb_shift_factor_received is a flag to indicate whether fgs_shifted_bits has been decoded in the previous fgs macroblock with the same fgs macroblock number. It is initialized to 0 before decoding the first vop-bp. It is set to 1 after fgs_shifted_bits is decoded.

NOTE 3 non_zero_macroblock is a flag to indicate whether every block-bp in the fgs macroblock is an all-zero block-bp. If every block-bp in the fgs macroblock is an all-zero block-bp, the value of this flag is 0. Otherwise, its value is 1.

NOTE 4 dct_type_received is a flag to indicate whether fgs_dct_type has been decoded in the previous fgs macroblock with the same fgs macroblock number. It is initialized to 0 before decoding the first vop-bp. It is set to 1 after fgs_dct_type is decoded.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.14.6 FGS Block

fgs_block() {	No. of bits	Mnemonic
if (fgs_vop_bp_id>=2 && previous_fgs_msb_not_reached==1)		
fgs_msb_not_reached	1	bslbf
if (fgs_msb_not_reached == 0) {		
while (eop == 0) {		
fgs_run_eop_code	1-16	vlclbf
if (coeff_msb_not_reached ==1)		
fgs_sign_bit	1	bslbf
}		
}		
}		

NOTE 1 previous_fgs_msb_not_reached is defined to be fgs_msb_not_reached decoded in the previous block-bp of the same 8x8 DCT block.

NOTE 2 eop is defined to be the EOP flag resulting from decoding the most recent fgs_run_eop_code. eop is reset to 0 at the beginning of fgs_block().

NOTE 3 coeff_msb_not_reached is defined to be an internal flag indicating, with a value '1', that the MSB of the non-zero DCT coefficient associated with the fgs_run_eop_code above was not reached. The value of this flag is changed to '0' when the MSB of the non-zero DCT coefficient is reached.

6.3 Visual bitstream semantics

6.3.1 Semantic rules for higher syntactic structures

This subclause details the rules that govern the way in which the higher level syntactic elements may be combined together to produce a legal bitstream. Subsequent subclauses detail the semantic meaning of all fields in the video bitstream.

Only when profile_and_level_indication indicates a studio profile, extension layers can be inserted in the bitstream. At each point where extensions are allowed in the bitstream any number of the extensions from the defined allowable set may be included. However each type of extension shall not occur more than once.

In the case that a decoder encounters an extension with an extension identification that is described as "reserved" in this specification the decoder shall discard all subsequent data until the next start code. This requirement allows future definition of compatible extensions to this specification.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-4 — extension_start_code_identifier codes

extension_start_code_identifier	Name
0000	reserved
0001	reserved
0010	Sequence Display Extension ID
0011	Quant Matrix Extension ID
0100	Copyright Extension ID
0101	VLC code Extension ID
0110	reserved
0111	Picture Display Extension ID
1000 - 1010	reserved
1011	Camera Parameters Extension ID
1100	ITU-T extension ID
1101	reserved
…	…
1111	reserved

6.3.2 Visual Object Sequence and Visual Object

visual_object_sequence_start_code: The visual_object_sequence_start_code is the bit string '000001B0' in hexadecimal. It initiates a visual session.

profile_and_level_indication: This is an 8-bit integer used to signal the profile and level identification. The meaning of the bits is given in Table G.1.

visual_object_sequence_end_code: The visual_object_sequence_end_code is the bit string '000001B1' in hexadecimal. It terminates a visual session.

visual_object_start_code: The visual_object_start_code is the bit string '000001B5' in hexadecimal. It initiates a visual object.

is_visual_object_identifier: This is a 1-bit code which when set to '1' indicates that version identification and priority is specified for the visual object. When set to '0', no version identification or priority needs to be specified.

visual_object_verid: This is a 4-bit code which identifies the version number of the visual object. Its meaning is defined in Table 6-5. When this field does not exist, the value of visual_object_verid is '0001'.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-5 — Meaning of visual_object_verid

visual_object_verid	Meaning
0000	reserved
0001	object type listed in Table 9-1
0010	object type listed in Table 9-2
0011	reserved
0100	object type listed in Table 9-3
0101	object type listed in Table 9-4
0110 - 1111	reserved

visual_object_priority: This is a 3-bit code which specifies the priority of the visual object. It takes values between 1 and 7, with 1 representing the highest priority and 7, the lowest priority. The value of zero is reserved.

visual_object_type: The visual_object_type is a 4-bit code given in Table 6-6 which identifies the type of the visual object.

Table 6-6 — Meaning of visual object type

code	visual object type
0000	reserved
0001	video ID
0010	still texture ID
0011	mesh ID
0100	FBA ID
0101	3D mesh ID
01101	reserved
:	:
:	:
1111	reserved

video_object_start_code: The video_object_start_code is a string of 32 bits. The first 27 bits are '0000 0000 0000 0000 0000 0001 000' in binary and the last 5-bits represent one of the values in the range of '00000' to '11111' in binary. The video_object_start_code marks a new video object.

video_object_id: This is given by the last 5-bits of the video_object_start_code. The video_object_id uniquely identifies a video object.

video_signal_type: A flag which if set to '1' indicates the presence of video_signal_type information.

video_format: This is a three bit integer indicating the representation of the pictures before being coded in accordance with this part of ISO/IEC 14496. Its meaning is defined in Table 6-7. If the video_signal_type() is not present in the bitstream then the video format may be assumed to be "Unspecified video format".

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-7 — Meaning of video_format

video_format	Meaning
000	Component
001	PAL
010	NTSC
011	SECAM
100	MAC
101	Unspecified video format
110	Reserved
111	Reserved

video_range: This one-bit flag indicates the black level and range of the luminance and chrominance signals.

colour_description: A flag which if set to '1' indicates the presence of colour_primaries, transfer_characteristics and matrix_coefficients in the bitstream.

colour_primaries: This 8-bit integer describes the chromaticity coordinates of the source primaries, and is defined in Table 6-8.

Table 6-8 — Colour Primaries

Value	Primaries		
0	(forbidden)		
1	ITU-R Recommendation BT.709		
	primary	x	y
	green	0,300	0,600
	blue	0,150	0,060
	red	0,640	0,330
	white D65	0,3127	0,3290
2	Unspecified Video Image characteristics are unknown.		
3	Reserved		
4	ITU-R Recommendation BT.470-2 System M		
	primary	x	y
	green	0,21	0,71
	blue	0,14	0,08
	red	0,67	0,33
	white C	0,310	0,316
5	ITU-R Recommendation BT.470-2 System B, G		
	primary	x	y
	green	0,29	0,60
	blue	0,15	0,06
	red	0,64	0,33
	white D65	0,3127	0,3290
6	SMPTE 170M		
	primary	x	y
	green	0,310	0,595
	blue	0,155	0,070
	red	0,630	0,340
	white D65	0,3127	0,3290

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2004 – All rights reserved
Not for Resale

7	SMPTE 240M (1987)		
	primary	x	y
	green	0,310	0,595
	blue	0,155	0,070
	red	0,630	0,340
	white D65	0,3127	0,3290
8	Generic film (colour filters using Illuminant C)		
	primary	x	y
	green	0,243	0,692 (Wratten 58)
	blue	0,145	0,049 (Wratten 47)
	red	0,681	0,319 (Wratten 25)
9-255	Reserved		

In the case that video_signal_type() is not present in the bitstream or colour_description is zero the chromaticity is assumed to be that corresponding to colour_primaries having the value 1.

transfer_characteristics: This 8-bit integer describes the opto-electronic transfer characteristic of the source picture, and is defined in Table 6-9.

Table 6-9 — Transfer Characteristics

Value	Transfer Characteristic
0	(forbidden)
1	ITU-R Recommendation BT.709 $V = 1{,}099\ L_C^{0{,}45} - 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	ITU-R Recommendation BT.470-2 System M Assumed display gamma 2,2
5	ITU-R Recommendation BT.470-2 System B, G Assumed display gamma 2,8
6	SMPTE 170M $V = 1{,}099\ L_C^{0{,}45} - 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
7	SMPTE 240M (1987) $V = 1{,}1115\ L_C^{0{,}45} - 0{,}1115$ for $L_C \geq 0{,}0228$ $V = 4{,}0\ L_C$ for $0{,}0228 > L_C$
8	Linear transfer characteristics i.e. $V = L_C$

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9	Logarithmic transfer characteristic (100:1 range) $V = 1.0-Log_{10}(Lc)/2$ for $1 = L_C = 0.01$ $V= 0.0$ for $0.01 > L_C$
10	Logarithmic transfer characteristic (316.22777:1 range) $V = 1.0-Log_{10}(Lc)/2.5$ for $1 = L_C = 0.0031622777$ $V= 0.0$ for $0.0031622777 > L_C$
11-255	reserved

In the case that video_signal_type() is not present in the bitstream or colour_description is zero the transfer characteristics are assumed to be those corresponding to transfer_characteristics having the value 1.

matrix_coefficients: This 8-bit integer describes the matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries, and is defined in Table 6-10.

In this table:

E'_Y is analogue with values between 0 and 1

E'_{PB} and E'_{PR} are analogue between the values -0,5 and 0,5

E'_R, E'_G and E'_B are analogue with values between 0 and 1

White is defined as $E'_y = 1$, $E'_{PB} = 0$, $E'_{PR} = 0$; $E'_R = E'_G = E'_B = 1$.

Y, Cb and Cr are related to E'_Y, E'_{PB} and E'_{PR} by the following formulae:

if **video_range=0:**

$Y = (219 * 2^{n-8} * E'_Y) + 2^{n-4}$.

$Cb = (224 * 2^{n-8} * E'_{PB}) + 2^{n-1}$

$Cr = (224 * 2^{n-8} * E'_{PR}) + 2^{n-1}$

if **video_range=1:**

$Y = ((2^n -1) * E'_Y)$

$Cb = ((2^n -1) * E'_{PB}) + 2^{n-1}$

$Cr = ((2^n -1) * E'_{PR}) + 2^{n-1}$

for n bit video.

For example, for 8 bit video,

video_range=0 gives a range of Y from 16 to 235, Cb and Cr from 16 to 240;

video_range=1 gives a range of Y from 0 to 255, Cb and Cr from 0 to 255.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-10 — Matrix Coefficients

Value	Matrix
0	(forbidden)
1	ITU-R Recommendation BT.709 $E'_Y = 0{,}7152\ E'_G + 0{,}0722\ E'_B + 0{,}2126\ E'_R$ $E'_{PB} = -0{,}386\ E'_G + 0{,}500\ E'_B - 0{,}115\ E'_R$ $E'_{PR} = -0{,}454\ E'_G - 0{,}046\ E'_B + 0{,}500\ E'_R$
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	FCC $E'_Y = 0{,}59\ E'_G + 0{,}11\ E'_B + 0{,}30\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}421\ E'_G - 0{,}079\ E'_B + 0{,}500\ E'_R$
5	ITU-R Recommendation BT.470-2 System B, G $E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$
6	SMPTE 170M $E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$
7	SMPTE 240M (1987) $E'_Y = 0{,}701\ E'_G + 0{,}087\ E'_B + 0{,}212\ E'_R$ $E'_{PB} = -0{,}384\ E'_G + 0{,}500\ E'_B - 0{,}116\ E'_R$ $E'_{PR} = -0{,}445\ E'_G - 0{,}055\ E'_B + 0{,}500\ E'_R$
8-255	reserved

In the case that video_signal_type() is not present in the bitstream or colour_description is zero the matrix coefficients are assumed to be those corresponding to matrix_coefficients having the value 1.

In the case that video_signal_type() is not present in the bitstream, video_range is assumed to have the value 0 (a range of Y from 16 to 235 for 8-bit video).

6.3.2.6 User data

user_data_start_code: The user_data_start_code is the bit string ‘000001B2’ in hexadecimal. It identifies the beginning of user data. The user data continues until receipt of another start code.

user_data: This is an 8 bit integer, an arbitrary number of which may follow one another. User data is defined by users for their specific applications. In the series of consecutive user_data bytes there shall not be a string of 23 or more consecutive zero bits.

6.3.3 Video Object Layer

video_object_layer_start_code: The video_object_layer_start_code is a string of 32 bits. The first 28 bits are ‘0000 0000 0000 0000 0000 0001 0010‘ in binary and the last 4-bits represent one of the values in the range of ‘0000’ to ‘1111’ in binary. The video_object_layer_start_code marks a new video object layer.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

video_object_layer_id: This is given by the last 4-bits of the video_object_layer_start_code. The video_object_layer_id uniquely identifies a video object layer.

short_video_header: The short_video_header is an internal flag which is set to 1 when an abbreviated header format is used for the VOP. This indicates video data which begins with a short_video_start_marker rather than a longer start code such as visual_object_ start_code. The short header format is included herein to provide forward compatibility with video codecs designed using the earlier video coding specification ITU-T Recommendation H.263. All decoders which support video objects shall support both header formats (short_video_header equal to 0 or 1) for the subset of video tools that is expressible in either form.

video_plane_with_short_header(): This is a syntax layer encapsulating a video plane which has only the limited set of capabilities available using the short header format.

random_accessible_vol: This flag may be set to "1" to indicate that every VOP in this VOL is individually decodable. If all of the VOPs in this VOL are intra-coded VOPs and some more conditions are satisfied then random_accessible_vol may be set to "1". The flag random_accessible_vol is not used by the decoding process. random_accessible_vol is intended to aid random access or editing capability. This shall be set to "0" if any of the VOPs in the VOL are non-intra coded or certain other conditions are not fulfilled.

video_object_type_indication: Constrains the following bitstream to use tools from the indicated object type only, e.g. Simple Object or Core Object, as shown in Table 6-11.

Table 6-11 — FLC table for video_object_type indication

Video Object Type	Code
Reserved	00000000
Simple Object Type	00000001
Simple Scalable Object Type	00000010
Core Object Type	00000011
Main Object Type	00000100
N-bit Object Type	00000101
Basic Anim. 2D Texture	00000110
Anim. 2D Mesh	00000111
Simple Face	00001000
Still Scalable Texture	00001001
Advanced Real Time Simple	00001010
Core Scalable	00001011
Advanced Coding Efficiency	00001100
Advanced Scalable Texture	00001101
Simple FBA	00001110
Simple Studio	00001111
Core Studio	00010000
Advanced Simple	00010001
Fine Granularity Scalable	00010010
Reserved	00010011 - 11111111

fgs_layer_type – This is a 2-bit code indicating whether this layer is FGS only, FGST only, or a combination of FGS and FGST. Table 6-12 shows the codes and the meanings.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-12 — Code for fgs_layer_type

Code	Meaning
00	reserved
01	FGS
10	FGST
11	FGS-FGST

is_object_layer_identifier: This is a 1-bit code which when set to '1' indicates that version identification and priority is specified for the visual object layer. When set to '0', no version identification or priority needs to be specified.

video_object_layer_verid: This is a 4-bit code which identifies the version number of the video object layer. Its meaning is defined in Table 6-13. If both visual_object_verid and video_object_layer_verid exist, the semantics of video_object_layer_verid supersedes the other. When this field does not exist, the value of video_object_layer_verid is substituted by the value of visual_object_verid.

Table 6-13 — Meaning of video_object_layer_verid

video_object_layer_verid	Meaning
0000	Reserved
0001	object type listed in Table 9-1
0010	object type listed in Table 9-2
0011	reserved
0100	object type listed in Table 9-3
0101	object type listed in Table 9-4
0110 - 1111	reserved

video_object_layer_priority: This is a 3-bit code which specifies the priority of the video object layer. It takes values between 1 and 7, with 1 representing the highest priority and 7 the lowest priority. The value of zero is reserved. For the transmission of FGS and FGST in two VOLs, the relative transmission priority of an FGS VOL vs. that of an FGST VOL can be specified by setting this parameter in the FGS VOL relative to the same parameter in the FGST VOL.

aspect_ratio_info: This is a four-bit integer which defines the value of pixel aspect ratio. Table 6-14 shows the meaning of the code. If aspect_ratio_info indicates extended PAR, pixel_aspect_ratio is represented by par_width and par_height. The par_width and par_height shall be relatively prime.

Table 6-14 — Meaning of pixel aspect ratio

aspect_ratio_info	pixel aspect ratios
0000	Forbidden
0001	1:1 (Square)
0010	12:11 (625-type for 4:3 picture)
0011	10:11 (525-type for 4:3 picture)
0100	16:11 (625-type stretched for 16:9 picture)
0101	40:33 (525-type stretched for 16:9 picture)
0110-1110	Reserved
1111	extended PAR

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

par_width: This is an 8-bit unsigned integer which indicates the horizontal size of pixel aspect ratio. A zero value is forbidden.

par_height: This is an 8-bit unsigned integer which indicates the vertical size of pixel aspect ratio. A zero value is forbidden.

vol_control_parameters: This a one-bit flag which when set to '1' indicates presence of the following parameters: chroma_format, low_delay, and vbv_parameters.

chroma_format: This is a two bit integer indicating the chrominance format as defined in the Table 6-15.

Table 6-15 — Meaning of chroma_format

chroma_format	Meaning
00	reserved
01	4:2:0
10	reserved
11	reserved

low_delay: This is a one-bit flag which when set to '1' indicates the VOL contains no B-VOPs. If this flag is not present in the bitstream, the default value is 0 for visual object types that support B-VOP otherwise it is 1.

vbv_parameters: This is a one-bit flag which when set to '1' indicates presence of following VBV parameters: first_half_bit_rate, latter_half_bit_rate, first_half_vbv_buffer_size, latter_half_vbv_buffer_size, first_half_vbv_occupancy and latter_half_vbv_occupancy. The VBV constraint is defined in Annex D.

first_half_bit_rate, latter_half_bit_rate: The bit rate is a 30-bit unsigned integer which specifies the bitrate of the bitstream measured in units of 400 bits/second, rounded upwards. The value zero is forbidden. This value is divided to two parts. The most significant bits are in first_half_bit_rate (15 bits) and the least significant bits are in latter_half_bit_rate (15 bits). The marker_bit is inserted between the first_half_bit_rate and the latter_half_bit_rate in order to avoid the resync_marker emulation. The instantaneous video object layer channel bit rate seen by the encoder is denoted by $R_{vol}(t)$ in bits per second. If the bit_rate (i.e. first_half_bit_rate and latter_half_bit_rate) field in the VOL header is present, it defines a peak rate (in units of 400 bits per second; a value of 0 is forbidden) such that $R_{vol}(t) <= 400 \times bit_rate$ Note that $R_{vol}(t)$ counts only visual syntax for the current elementary stream (also see Annex D).

first_half_vbv_buffer_size, latter_half_vbv_buffer_size: vbv_buffer_size is an 18-bit unsigned integer. This value is divided into two parts. The most significant bits are in first_half_vbv_buffer_size (15 bits) and the least significant bits are in latter_half_vbv_buffer_size (3 bits), The VBV buffer size is specified in units of 16384 bits. The value 0 for vbv_buffer_size is forbidden. Define B = 16384 × vbv_buffer_size to be the VBV buffer size in bits.

first_half_vbv_occupancy, latter_half_vbv_occupancy: The vbv_occupancy is a 26-bit unsigned integer. This value is divided to two parts. The most significant bits are in first_half_vbv_occupancy (11 bits) and the least significant bits are in latter_half_vbv_occupancy (15 bits). The marker_bit is inserted between the first_half_vbv_occupancy and the latter_half_vbv_occupancy in order to avoid the resync_marker emulation. The value of this integer is the VBV occupancy in 64-bit units just before the removal of the first VOP following the VOL header. The purpose for the quantity is to provide the initial condition for VBV buffer fullness.

video_object_layer_shape: This is a 2-bit integer defined in Table 6-16. It identifies the shape type of a video object layer.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-16 — Video Object Layer shape type

Shape format	Meaning
00	rectangular
01	binary
10	binary only
11	grayscale

If video_object_layer_shape == 'rectangular', no shape coding tool (i.e. binary shape or grey scale shape) is used to decode the video object layer. In such a VOL, all macroblocks of each VOP contain the same number of 8x8 blocks, even when (video_object_layer_height % 16) ≤ 8 or (video_object_layer_width % 16) ≤ 8 would make fewer blocks per macroblock possible by using a shape-coding tool for some macroblocks.

video_object_layer_shape_extension: This is a 4-bit integer defined in Table 6-17. It identifies the number (up to 3) and type of auxiliary components that can be used, including the grayscale shape (ALPHA) component. Only a limited number of types and combinations are defined in Table 6-17. More applications are possible by selection of the USER DEFINED type.

Table 6-17 — Semantic meaning of video_object_layer_shape_extension

video_ object_ layer_ shape_ extension	aux_comp_type[0]	aux_comp_type[1]	aux_comp_type[2]	aux_ comp_ count
0000	ALPHA	NO	NO	1
0001	DISPARITY	NO	NO	1
0010	ALPHA	DISPARITY	NO	2
0011	DISPARITY	DISPARITY	NO	2
0100	ALPHA	DISPARITY	DISPARITY	3
0101	DEPTH	NO	NO	1
0110	ALPHA	DEPTH	NO	2
0111	TEXTURE	NO	NO	1
1000	USER DEFINED	NO	NO	1
1001	USER DEFINED	USER DEFINED	NO	2
1010	USER DEFINED	USER DEFINED	USER DEFINED	3
1011	ALPHA	USER DEFINED	NO	2
1100	ALPHA	USER DEFINED	USER DEFINED	3
1101-1111	t.b.d.	t.b.d.	t.b.d.	t.b.d.

vop_time_increment_resolution: This is a 16-bit unsigned integer that indicates the number of evenly spaced subintervals, called ticks, within one modulo time. One modulo time represents the fixed interval of one second. The value zero is forbidden.

fixed_vop_rate: This is a one-bit flag which indicates that all VOPs are coded with a fixed VOP rate. It shall only be '1' if and only if all the distances between the display time of any two successive VOPs in the display order in the video object layer are constant. In this case, the VOP rate can be derived from the fixed_vop_time_increment. If it is '0' the display time between any two successive VOPs in the display order can be variable thus indicated by the time stamps provided in the VOP header.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fixed_vop_time_increment: This value represents the number of ticks between two successive VOPs in the display order. The length of a tick is given by vop_time_increment_resolution. It can take a value in the range of [0,vop_time_increment_resolution). The number of bits representing the value is calculated as the minimum number of unsigned integer bits required to represent the above range. fixed_vop_time_increment shall only be present if fixed_vop_rate is '1' and its value must be identical to the constant given by the distance between the display time of any two successive VOPs in the display order. In this case, the fixed VOP rate is given as (vop_time_increment_resolution / fixed_vop_time_increment). A zero value is forbidden.

EXAMPLE
VOP time = tick × vop_time_increment
= vop_time_increment / vop_time_increment_resolution

Table 6-18 — Examples of vop_time_increment_resolution, fix_vop_time_increment, and vop_time_increment

Fixed VOP rate = 1/VOP time	vop_time_increment_ resolution	fixed_vop_time_ increment	vop_time_increment
15Hz	15	1	0, 1, 2, 3, 4,…
7.5Hz	15	2	0, 2, 4, 6, 8,…
29.97…Hz	30000	1001	0, 1001, 2002, 3003,…
59.94…Hz	60000	1001	0, 1001, 2002, 3003,…

video_object_layer_width: The video_object_layer_width is a 13-bit unsigned integer representing the width of the displayable part of the luminance component in pixel units. The width of the encoded luminance component of VOPs in macroblocks is (video_object_layer_width+15)/16. The displayable part is left-aligned in the encoded VOPs. A zero value is forbidden.

video_object_layer_height: The video_object_layer_height is a 13-bit unsigned integer representing the height of the displayable part of the luminance component in pixel units. The height of the encoded luminance component of VOPs in macroblocks is (video_object_layer_height+15)/16. The displayable part is top-aligned in the encoded VOPs. A zero value is forbidden.

interlaced: This is a 1 bit flag which, when set to “1” indicates that the VOPs associated with the VOL may contain interlaced video. When this flag is set to “0”, the VOPs associated with the VOL are of non-interlaced (or progressive) format.

fgs_ref_layer_id: This is a 4-bit unsigned integer with value between 0 and 15. It indicates the layer to be used as reference for prediction in the case of fgs_layer_type being FGST or FGS_FGST.

fgs_frequency_weighting_enable: This is a one-bit flag to indicate that frequency weighting is used in this VOL, when set to ‘1’. Otherwise, when this flag is set to ‘0’, frequency weighting is not used. The default frequency weighting matrix is an all zero matrix when fgs_frequency_weighting_enable is ‘1’.

load_fgs_frequency_weighting_matrix: This is a one-bit flag which is set to ‘1’ when fgs_frequency_weighting_matrix follows. If it is set to ‘0’ then the default frequency weighting matrix is used.

fgs_frequency_weighting_matrix: This is a list of 2 to 64 three-bit unsigned integers. The integers are in zigzag scan order representing the fgs_frequency_weighting_matrix. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set to zero.

fgst_frequency_weighting_enable: This is a one-bit flag to indicate that frequency weighting is used in this VOL, when set to ‘1’. Otherwise, when this flag is set to ‘0’, frequency weighting is not used. The default frequency weighting matrix is an all zero matrix when fgst_frequency_weighting_enable is ‘1’.

load_fgst_frequency_weighting_matrix: This is a one-bit flag which is set to ‘1’ when fgst_frequency_weighting_matrix follows. If it is set to ‘0’ then the default matrix is used.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fgst_frequency_weighting_matrix: This is a list of 2 to 64 three-bit unsigned integers. The integers are in zigzag scan order representing the fgst_frequency_weighting_matrix. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set to zero.

fgs_resync_marker_disable: This is a one-bit flag which when set to '1' indicates that there is no fgs_resync_marker in coded fgs vops of this VOL. When this flag is set to '0', it indicates that fgs_resync_marker may be used in coded fgs vops of this VOL.

obmc_disable: This is a one-bit flag which when set to '1' disables overlapped block motion compensation.

sprite_enable: When video_object_layer_verid == '0001', this is a one-bit flag which when set to '1' indicates the usage of static (basic or low latency) sprite coding. When video_object_layer_verid == '0010' or video_object_layer_verid == '0101', this is a two-bit unsigned integer which indicates the usage of static sprite coding or global motion compensation (GMC). Table 6-19 shows the meaning of various codewords. An S-VOP with sprite_enable == "GMC" is referred to as an S (GMC)-VOP in this document.

Table 6-19 — Meaning of sprite_enable codewords

sprite_enable (video_object_layer_ verid == '0001')	sprite_enable (video_object_layer_ verid == '0010') || video_object_layer_verid == '0101')	Sprite Coding Mode
0	00	sprite not used
1	01	static (Basic/Low Latency)
–	10	GMC (Global Motion Compensation)
–	11	Reserved

sprite_width: This is a 13-bit unsigned integer which identifies the horizontal dimension of the sprite.

sprite_height: This is a 13-bit unsigned integer which identifies the vertical dimension of the sprite.

sprite_left_coordinate: This is a 13-bit signed integer which defines the left edge of the sprite. The value of sprite_left_coordinate shall be divisible by two.

sprite_top_coordinate: This is a 13-bit signed integer which defines the top edge of the sprite. The value of sprite_top_coordinate shall be divisible by two.

no_of_sprite_warping_points: This is a 6-bit unsigned integer which represents the number of points used in sprite warping. When its value is 0 and when sprite_enable is set to 'static' or 'GMC', warping is identity (stationary sprite) and no coordinates need to be coded. When its value is 4, a perspective transform is used. When its value is 1,2 or 3, an affine transform is used. Further, the case of value 1 is separated as a special case from that of values 2 or 3. Table 6-20 shows the various choices. Note that the value of 4 is disallowed when sprite_enable == 'GMC'.

Table 6-20 — Number of point and implied warping function

Number of points	warping function
0	Stationary
1	Translation
2,3	Affine
4	Perspective
5-63	Reserved

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sprite_warping_accuracy: This is a 2-bit code which indicates the quantisation accuracy of motion vectors used in the warping process for sprites and GMC. Table 6-21 shows the meaning of various codewords

Table 6-21 — Meaning of sprite warping accuracy codewords

code	sprite_warping_accuracy
00	1/2 pixel
01	1/4 pixel
10	1/8 pixel
11	1/16 pixel

sprite_brightness_change: This is a one-bit flag which when set to '1' indicates a change in brightness during sprite warping, alternatively, a value of '0' means no change in brightness. This value shall be '0' when sprite_enable=='GMC'.

low_latency_sprite_enable: This is a one-bit flag which when set to "1" indicates the presence of low_latency sprite, alternatively, a value of "0" means basic sprite.

not_8_bit: This one bit flag is set when the video data precision is not 8 bits per pixel and visual object type is N-bit.

sadct_disable: This is a one-bit flag specifying the inverse transforms to be used for texture decoding. If 'sadct_disable' is set to '1', inverse DCT as described in version 1 is applied to all 8x8-blocks. When set to '0', flag 'sadct_disable' indicates that different types of inverse DCT are used in an adaptive way: inverse DCT is applied to those 8x8-blocks where all 64 pels are opaque, whereas inverse shape-adaptive DCT (SA-DCT) and inverse ΔDC-SA-DCT – an extended version of SA-DCT - are used in inter- and intra-coded 8x8-blocks with at least one transparent and one opaque pel.

quant_precision: This field specifies the number of bits used to represent quantiser scales. Values between 3 and 9 are allowed. When not_8_bit is zero, and therefore quant_precision is not transmitted, it takes a default value of 5.

bits_per_pixel: This field specifies the video data precision in bits per pixel. It may take different values for different video object layers within a single video object. A value of 12 in this field would indicate 12 bits per pixel. This field may take values between 4 and 12. When not_8_bit is zero and bits_per_pixel is not present, the video data precision is always 8 bits per pixel, which is equivalent to specifying a value of 8 in this field. The same number of bits per pixel is used in the luminance and two chrominance planes. The alpha plane, used to specify shape of video objects, is always represented with 8 bits per pixel.

no_gray_quant_update: This is a one bit flag which is set to '1' when a fixed quantiser scale is used for the decoding of grayscale alpha data. When this flag is set to '0', the grayscale alpha quantiser scale is updated on every macroblock by generating it anew from the luminance quantiser scale value, but with an appropriate scale factor applied. See the description in subclause 7.5.4.3.

composition_method: This is a one bit flag which indicates which blending method is to be applied to the video object in the compositor. When set to '0', cross-fading shall be used. When set to '1', additive mixing shall be used. See subclause 7.5.4.6.

linear_composition: This is a one bit flag which indicates the type of signal used by the compositing process. When set to '0', the video signal in the format from which it was produced by the video decoder is used. When set to '1', linear signals are used. See subclause 7.5.4.6.

quant_type: This is a one-bit flag which when set to '1' that the first inverse quantisation method and when set to '0' indicates that the second inverse quantisation method is used for inverse quantisation of the DCT coefficients. Both inverse quantisation methods are described in subclause 7.4.4. For the first inverse quantisation method, two matrices are used, one for intra blocks the other for non-intra blocks.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The default matrix for intra blocks is:

8	17	18	19	21	23	25	27
17	18	19	21	23	25	27	28
20	21	22	23	24	26	28	30
21	22	23	24	26	28	30	32
22	23	24	26	28	30	32	35
23	24	26	28	30	32	35	38
25	26	28	30	32	35	38	41
27	28	30	32	35	38	41	45

The default matrix for non-intra blocks is:

16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24
18	19	20	21	22	23	24	25
19	20	21	22	23	24	26	27
20	21	22	23	25	26	27	28
21	22	23	24	26	27	28	30
22	23	24	26	27	28	30	31
23	24	25	27	28	30	31	33

load_intra_quant_mat: This is a one-bit flag which is set to ‘1’ when intra_quant_mat follows. If it is set to ‘0’ then there is no change in the values that shall be used.

intra_quant_mat: This is a list of 2 to 64 eight-bit unsigned integers. The new values are in zigzag scan order and replace the previous values. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set equal to the last non-zero value. The first value shall always be 8 and is not used in the decoding process.

load_nonintra_quant_mat: This is a one-bit flag which is set to ‘1’ when nonintra_quant_mat follows. If it is set to ‘0’ then there is no change in the values that shall be used.

nonintra_quant_mat: This is a list of 2 to 64 eight-bit unsigned integers. The new values are in zigzag scan order and replace the previous values. A value of 0 indicates that no more values are transmitted and the remaining, non-transmitted values are set equal to the last non-zero value. The first value shall not be 0.

load_intra_quant_mat_grayscale: This is a one-bit flag which is set to ‘1’ when intra_quant_mat_grayscale follows. If it is set to ‘0’ then there is no change in the quantisation matrix values that shall be used.

intra_quant_mat_grayscale: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale intra alpha quantisation matrix to be used. The semantics and the default quantisation matrix are identical to those of intra_quant_mat.

load_nonintra_quant_mat_grayscale: This is a one-bit flag which is set to ‘1’ when nonintra_quant_mat_grayscale[i] follows for grayscale alpha or auxiliary component i=0,1,2. If it is set to ‘0’ then there is no change in the quantisation matrix values that shall be used.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

intra_quant_mat_grayscale[i]: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale intra alpha quantisation matrix to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those of intra_quant_mat.

nonintra_quant_mat_grayscale: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix[i] to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those of nonintra_quant_mat.

nonintra_quant_mat_grayscale[i]: This is a list of 2 to 64 eight-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix to be used for grayscale alpha or auxiliary component i=0,1,2. The semantics and the default quantisation matrix are identical to those of nonintra_quant_mat.

quarter_sample: This is a one-bit flag which when set to ‘0’ indicates that half sample mode and when set to ‘1’ indicates that quarter sample mode shall be used for motion compensation of the luminance component. For FGST or FGS_FGST enhancement layer, this flag shall be 0.

complexity_estimation_disable: This is a one-bit flag which, when set to '1', disables complexity estimation header in each VOP.

estimation_method: Setting of the of the estimation method, it is „00“ for Version 1 and “01” for version 2.

shape_complexity_estimation_disable: This is a one-bit flag which when set to '1' disables shape complexity estimation.

opaque: Flag enabling transmission of the number of luminance and chrominance blocks coded using opaque coding mode in % of the total number of blocks (bounding rectangle).

transparent: Flag enabling transmission of the number of luminance and chrominance blocks coded using transparent mode in % of the total number of blocks (bounding rectangle).

intra_cae: Flag enabling transmission of the number of luminance and chrominance blocks coded using IntraCAE coding mode in % of the total number of blocks (bounding rectangle).

inter_cae: Flag enabling transmission of the number of luminance and chrominance blocks coded using InterCAE coding mode in % of the total number of blocks (bounding rectangle).

no_update: Flag enabling transmission of the number of luminance and chrominance blocks coded using no update coding mode in % of the total number of blocks (bounding rectangle).

upsampling: Flag enabling transmission of the number of luminance and chrominance blocks which need upsampling from 4-4- to 8-8 block dimensions in % of the total number of blocks (bounding rectangle).

version2_complexity_estimation_disable: Flag to disable version 2 parameter set.

sadct: Flag enabling transmission of the number of luminance and chrominance blocks coded using SADCT coding mode in % of the total number of blocks (bounding box). When estimation_method == ‘00’ the value of sadct is set to ‘0’.

quarterpel: Flag enabling transmission of the number of luminance and chrominance block predicted by a quarter-pel vector on one or two dimensions (horizontal and vertical) in % of the total number of blocks (bounding box). When estimation_method == ‘00’ the value of quarterpel is set to ‘0’.

texture_complexity_estimation_set_1_disable: Flag to disable texture parameter set 1.

intra_blocks: Flag enabling transmission of the number of luminance and chrominance Intra or Intra+Q coded blocks in % of the total number of blocks (bounding rectangle).

inter_blocks: Flag enabling transmission of the number of luminance and chrominance Inter and Inter+Q coded blocks in % of the total number of blocks (bounding rectangle).

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

inter4v_blocks: Flag enabling transmission of the number of luminance and chrominance Inter4V coded blocks in % of the total number of blocks (bounding rectangle).

not_coded_blocks: Flag enabling transmission of the number of luminance and chrominance Non Coded blocks in % of the total number of blocks (bounding rectangle).

texture_complexity_estimation_set_2_disable: Flag to disable texture parameter set 2.

dct_coefs: Flag enabling transmission of the number of DCT coefficients % of the maximum number of coefficients (coded blocks).

dct_lines: Flag enabling transmission of the number of DCT8x1 in % of the maximum number of DCT8x1 (coded blocks).

vlc_symbols: Flag enabling transmission of the average number of VLC symbols for macroblock.

vlc_bits: Flag enabling transmission of the average number of bits for each symbol.

motion_compensation_complexity_disable: Flag to disable motion compensation parameter set.

apm (Advanced Prediction Mode): Flag enabling transmission of the number of luminance block predicted using APM in % of the total number of blocks for VOP (bounding rectangle).

npm (Normal Prediction Mode): Flag enabling transmission of the number of luminance and chrominance blocks predicted using NPM in % of the total number of luminance and chrominance for VOP (bounding rectangle).

interpolate_mc_q: Flag enabling transmission of the number of luminance and chrominance interpolated blocks in % of the total number of blocks for VOP (bounding rectangle).

forw_back_mc_q: Flag enabling transmission of the number of luminance and chrominance predicted blocks in % of the total number of blocks for VOP (bounding rectangle).

halfpel2: Flag enabling transmission of the number of luminance and chrominance block predicted by a half-pel vector on one dimension (horizontal or vertical) in % of the total number of blocks (bounding rectangle).

halfpel4: Flag enabling transmission of the number of luminance and chrominance block predicted by a half-pel vector on two dimensions (horizontal and vertical) in % of the total number of blocks (bounding rectangle).

resync_marker_disable: This is a one-bit flag which when set to ‘1‘ indicates that there is no resync_marker in coded VOPs. This flag can be used only for the optimization of the decoder operation. Successful decoding can be carried out without taking into account the value of this flag.

data_partitioned: This is a one-bit flag which when set to ‘1’ indicates that the macroblock data is rearranged differently, specifically, motion vector data is separated from the texture data (i.e., DCT coefficients).

reversible_vlc: This is a one-bit flag which when set to ‘1’ indicates that the reversible variable length tables (Table B.23, Table B.24 and Table B.25) should be used when decoding DCT coefficients. These tables can only be used when data_partition flag is enabled. Note that this flag shall be treated as ‘0’ in B-VOPs. Use of escape sequence (Table B.24 and Table B.25) for encoding the combinations listed in Table B.23 is prohibited.

newpred_enable: This is a one-bit flag which, when set to ‘1’, indicates that the NEWPRED mode is enabled. When video_object_layer_verid is equal to ’0001’, and therefore newpred enable is not transmitted, it takes a default value of zero.

requested_upstream_message_type: This is a twe-bits flag which indicates which type of upstream message is needed by the encoder. The syntax and semantics of the upstream message are described in subclause 6.2.12 and 6.3.12.

-`,,`-`-`,,`,,`,`,,`-`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

01: need NP_ACK message to be returned for each NEWPRED segment
10: need NP_NACK message to be returned for each NEWPRED segment
11: need both NP_ACK and NP_NACK messages to be returned for each NEWPRED segment
00: reserved

newpred_segment_type: This is a one-bits flag which indicates the unit of selecting reference VOP (NEWPRED segment).

0: Video Packet
1: VOP

reduced_resolution_vop_enable: This is a one-bit flag which indicates that the reduced resolution VOP tool is enabled when set to '1'. When video_object_layer_verid is equal to '0001', and therefore reduced_resolution_vop_enable is not transmitted, it takes a default value of zero.

scalability: This is a one-bit flag which when set to '1' indicates that the current layer uses scalable coding. If the current layer is used as base-layer then this flag is set to '0'. Additionally, this flag shall be set to '0' for S(GMC)-VOPs.

hierarchy_type: The hierarcical relation between the associated hierarchy layer and its hierarchy embedded layer is defined as shown in Table 6-22.

Table 6-22 — Code table for hierarchy_type

Description	Code
ISO/IEC 14496-2 Spatial Scalability	0
ISO/IEC 14496-2 Temporal Scalability	1

ref_layer_id: This is a 4-bit unsigned integer with value between 0 and 15. It indicates the layer to be used as reference for prediction(s) in the case of scalability.

ref_layer_sampling_direc: This is a one-bit flag which when set to '1' indicates that the resolution of the reference layer (specified by reference_layer_id) is higher than the resolution of the layer being coded. If it is set to '0' then the reference layer has the same or lower resolution than the resolution of the layer being coded.

hor_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in horizontal spatial resampling in scalability. The value of zero is forbidden.

hor_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in horizontal spatial resampling in scalability. The value of zero is forbidden.

vert_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in vertical spatial resampling in scalability. The value of zero is forbidden.

vert_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in vertical spatial resampling in scalability. The value of zero is forbidden.

enhancement_type: This is a 1-bit flag which is set to '1' when the current layer enhances the partial region of the reference layer. If it is set to '0' then the current layer enhances the entire region of the reference layer. The default value of this flag is '0'.

use_ref_shape: This is a one bit flag which indicates the procedure to decode binary shape for spatial scalability. If it is set to '0', scalable shape coding should be used. If it is set to '1' and enhancement_type is set to '0', no shape data is decoded and the up-sampled binary shape of reference_layer should be used for enhancement layer. If enhancement_type is set to '1' and this flag is set to '1', binary shape of the enhancement layer should be decoded as the same non-scalable decoding process. When video_object_layer_verid == '0001', the value of use_ref_shape is set to '1'.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

use_ref_texture: Reserved flag for future extension. This flag shall be 0 when video_object_layer_ver_id is "0001" or "0010".

shape_hor_sampling_factor_n: This is a 5-bit unsigned integer which forms the numerator of the ratio used in horizontal spatial resampling in shape scalability. The value of zero is forbidden.

shape_hor_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in horizontal spatial resampling in shape scalability. The value of zero is forbidden.

shape_vert_sampling_factor_n: This is a 5-bit unsigned integer which forms the denominator of the ratio used in vertical spatial resampling in shape scalability. The value of zero is forbidden.

shape_vert_sampling_factor_m: This is a 5-bit unsigned integer which forms the denominator of the ratio used in vertical spatial resampling in shape scalability. The value of zero is forbidden.

stuffing_start_code: This is the bit string '000001C3' in hexadecimal. It is used in conjunction with possibly following stuffing_byte(s) for the purpose of stuffing bits to guaranty the VBV buffer regulation.

stuffing_byte: This is the 8-bit string in which the value is '11111111'.

6.3.4 Group of Video Object Plane

group_of_vop_start_code: The group_of_vop_start_code is the bit string '000001B3' in hexadecimal. It identifies the beginning of a GOV header.

time_code: This is a 18-bit integer containing the following: time_code_hours, time_code_minutes, marker_bit and time_code_seconds as shown in Table 6-23. The parameters correspond to those defined in the IEC standard publication 461 for "time and control codes for video tape recorders". The time code specifies the modulo part (i.e. the full second units) of the time base for the first object plane (in display order) after the GOV header.

Table 6-23 — Meaning of time_code

time_code	range of value	No. of bits	Mnemonic
time_code_hours	0 - 23	5	uimsbf
time_code_minutes	0 - 59	6	uimsbf
marker_bit	1	1	bslbf
time_code_seconds	0 - 59	6	uimsbf

closed_gov: This is a one-bit flag which indicates the nature of the predictions used in the first consecutive B-VOPs (if any) immediately following the first coded I-VOP after the GOV header .The closed_gov is set to '1' to indicate that these B-VOPs have been encoded using only backward prediction or intra coding. This bit is provided for use during any editing which occurs after encoding. If the previous pictures have been removed by editing, broken_link may be set to '1' so that a decoder may avoid displaying these B-VOPs following the first I-VOP following the group of plane header. However if the closed_gov bit is set to '1', then the editor may choose not to set the broken_link bit as these B-VOPs can be correctly decoded.

broken_link: This is a one-bit flag which shall be set to '0' during encoding. It is set to '1' to indicate that the first consecutive B-VOPs (if any) immediately following the first coded I-VOP following the group of plane header may not be correctly decoded because the reference frame which is used for prediction is not available (because of the action of editing). A decoder may use this flag to avoid displaying frames that cannot be correctly decoded.

6.3.5 Video Object Plane and Video Plane with Short Header

vop_start_code: This is the bit string '000001B6' in hexadecimal. It marks the start of a video object plane.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

vop_coding_type: The vop_coding_type identifies whether a VOP is an intra-coded VOP (I), predictive-coded VOP (P), bidirectionally predictive-coded VOP (B) or sprite coded VOP (S). The meaning of vop_coding_type is defined in Table 6-24.

Table 6-24 — Meaning of vop_coding_type

vop_coding_type	coding method
00	intra-coded (I)
01	predictive-coded (P)
10	bidirectionally-predictive-coded (B)
11	sprite (S)

modulo_time_base: This value represents the local time base in one second resolution units. It consists of a number of consecutive '1' followed by a '0'. Each '1' represents a duration of one second that have elapsed. For I-, S(GMC)-, and P-VOPs of a non scalable bitstream and the base layer of a scalable bitstream, the number of '1's indicate the number of seconds elapsed since the synchronization point marked by time_code of the previous GOV header or by non-zero modulo_time_base of the previously decoded I-, S(GMC)-, or P-VOP, in decoding order. For B-VOP of non scalable bitstream and base layer of scalable bitstream, the number of '1's indicate the number of seconds elapsed since the synchronization point marked in the previous GOV header, I-VOP, S(GMC)-VOP, or P-VOP, in display order. For I-, P-, or B-VOPs of enhancement layer of scalable bitstream, the number of '1's indicate the number of seconds elapsed since the synchronization point marked in the previous GOV header, I-VOP, P-VOP, or B-VOP, in display order.

vop_time_increment: This value represents the absolute vop_time_increment from the synchronization point marked by the modulo_time_base measured in the number of clock ticks. It can take a value in the range of [0,vop_time_increment_resolution). The number of bits representing the value is calculated as the minimum number of unsigned integer bits required to represent the above range, unless vop_time_increment_resolution is equal to '1', in which case one bit with a value of '0' is used. The local time base in the units of seconds is recovered by dividing this value by the vop_time_increment_resolution.

vop_coded: This is a 1-bit flag which when set to '0' indicates that no subsequent data exists for the VOP. In this case, the following decoding rules apply: If binary shape or alpha plane does exist for the VOP (i.e. video_object_layer_shape != "rectangular"), it shall be completely transparent. If binary shape or alpha plane does not exist for the VOP (i.e. video_object_layer_shape == "rectangular"), the luminance and chrominance planes of the reconstructed VOP shall be filled with the forward reference VOP as defined in subclause 7.6.7.

vop_rounding_type: This is a one-bit flag which signals the value of the parameter rounding_control used for pixel value interpolation in motion compensation for P- and S(GMC)- VOPs. When this flag is set to '0', the value of rounding_control is 0, and when this flag is set to '1', the value of rounding_control is 1. When vop_rounding_type is not present in the VOP header, the value of rounding_control is 0.

vop_reduced_resolution: This single bit flag signals whether the VOP is encoded in spatialy reduced resolution or not. When vop_reduced_resolution is not transmitted, it takes a default value of zero. When this flag is set to '1', the VOP is encoded spatialy reduced resolution and referred as Reduced Resolution VOP. Reduced Resolution VOP shall be decoded by the texture decoding process including upsampling of IDCT output, and motion compensation decoding process of Reduced Resolution VOP. In this case, the width of the encoded luminance component of the VOP in macroblocks is (video_object_layer_width+31)/32 and the height of the encoded luminance component of the VOP in macroblocks is (video_object_layer_height+31)/32. The displayable part is top and left-aligned in the encoded VOP. This flag shall not be the '1' when the (video_object_layer_width+15)/16 is not the multiple of 2 or the (video_object_layer_width+15)/16 is not the multiple of 2. When this flag is set to "0" or this flag is not present, the VOP is encoded in normal spatial resolution and shall be decoded by the normal decoding process.

vop_width: This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the VOP. The width of the encoded luminance component of VOP in macroblocks is (vop_width+15)/16. The rectangle part is left-aligned in the encoded VOP. A zero value is forbidden.

--`,,`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

vop_height: This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the VOP. The height of the encoded luminance component of VOP in macroblocks is (vop_height+15)/16. The rectangle part is top-aligned in the encoded VOP. A zero value is forbidden.

vop_horizontal_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle defined by horizontal size of vop_width. The value of vop_horizontal_mc_spatial_ref shall be divisible by two. This is used for decoding and for picture composition.

vop_vertical_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle defined by vertical size of vop_height. The value of vop_vertical_mc_spatial_ref shall be divisible by two for progressive and divisible by four for interlaced motion compensation. This is used for decoding and for picture composition.

background_composition: This flag only occurs when scalability flag has a value of "1. This flag is used in conjunction with enhancement_type flag. If enhancement_type is "1", hierarchy_type is "1", and this flag is "1", background composition specified in subclause 8.1 is performed. If enhancement type is "1", hierarchy_type is "1" and this flag is "0", any method can be used to make a background for the enhancement layer.

When hierarchy_type is "0", video_object_layer_shape is "binary" (object based spatial scalability), enhancement_type is "1", and background_compositin flag is "1", background composition specified in subclause 8.4 is performed and when enhancement_type is "1" and this flag is "0", any method can be used to make a background for the enhancement layer.

change_conv_ratio_disable: This is a 1-bit flag which when set to '1' indicates that conv_ratio is not sent at the macroblock layer and is assumed to be 1 for all the macroblocks of the VOP. When set to '0', the conv_ratio is coded at macroblock layer.

vop_constant_alpha: This bit is used to indicate the presence of vop_constant_alpha_value. When this is set to one, vop_constant_alpha_value is included in the bitstream.

vop_constant_alpha_value: This is an unsigned integer which indicates the scale factor to be applied as a post processing phase of binary or grayscale shape decoding. See subclause 7.5.4.2.

intra_dc_vlc_thr: This is a 3-bit code that specify a threshold value of quantiser scale used to switch between two VLC's for coding of Intra DC coefficients as per Table 6-25.

Table 6-25 — Meaning of intra_dc_vlc_thr

index	meaning of intra_dc_vlc_thr	code
0	Use Intra DC VLC for entire VOP	000
1	Switch to Intra AC VLC at running Qp >=13	001
2	Switch to Intra AC VLC at running Qp >=15	010
3	Switch to Intra AC VLC at running Qp >=17	011
4	Switch to Intra AC VLC at running Qp >=19	100
5	Switch to Intra AC VLC at running Qp >=21	101
6	Switch to Intra AC VLC at running Qp >=23	110
7	Use Intra AC VLC for entire VOP	111

Where running Qp is defined as the DCT quantiser scale for luminance and chrominance used for immediately previous coded macroblock, except for the first coded macroblock in a VOP or a video packet. At the first coded macroblock in a VOP or a video packet, the running Qp is defined as the quantiser scale value for the current macroblock. When usage of Intra DC VLC is specified, the intra DC coefficients are coded by dct_dc_size_luminance, dct_dc_size_chrominance and dct_dc_differential using Table B.13 to Table B.15. When usage of Intra AC VLC is specified, the intra DC coefficients are coded as intra AC coefficients using Table 6-20.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

top_field_first: This is a 1-bit flag which when set to “1” indicates that the top field (i.e., the field containing the top line) of reconstructed VOP is the first field to be displayed (output by the decoding process). When top_field_first is set to “0” it indicates that the bottom field of the reconstructed VOP is the first field to be displayed.

alternate_vertical_scan_flag: This is a 1-bit flag which when set to “1” indicates the use of alternate vertical scan for interlaced VOPs.

sprite_transmit_mode: This is a 2-bit code which signals the transmission mode of the sprite object. At video object layer initialization, the code is set to “piece” mode. When all object and quality update pieces are sent for the entire video object layer, the code is set to the “stop”mode. When an object piece is sent, the code is set to “piece” mode. When an update piece is being sent, the code is set to the “update” mode. When all sprite object pieces andquality update pieces for the current VOP are sent, the code is set to “pause” mode. Table 6-26 shows the different sprite transmit modes.

Table 6-26 — Meaning of sprite transmit modes

code	sprite_transmit_mode
00	stop
01	piece
10	update
11	pause

vop_quant: This is an unsigned integer which specifies the absolute value of quantiser scale to be used for inverse quantising the macroblock until updated by any subsequent dquant, dbquant, or quant_scale. The length of this field is specified by the value of the parameter quant_precision. The default length is 5-bits which carries the binary representation of quantiser scale values from 1 to 31 in steps of 1.

vop_alpha_quant[i]: This is a an unsigned integer which specifies the absolute value of the initial alpha plane quantiser scale to be used for inverse quantising macroblock grayscale alpha data in alpha or auxiliary component i=0,1,2. The alpha plane quantiser scale cannot be less than 1.

vop_fcode_forward: This is a 3-bit unsigned integer taking values from 1 to 7; the value of zero is forbidden. It is used in decoding of motion vectors.

vop_fcode_backward: This is a 3-bit unsigned integer taking values from 1 to 7; the value of zero is forbidden. It is used in decoding of motion vectors.

vop_shape_coding_type: This is a 1 bit flag which specifies whether inter shape decoding is to be carried out for the current P, B, or S(GMC)-VOP. If vop_shape_coding_type is equal to ‘0’, intra shape decoding is carried out, otherwise inter shape decoding is carried out.

Coded data for the top-left macroblock of the bounding rectangle of a VOP shall immediately follow the VOP header, followed by the remaining macroblocks in the bounding rectangle in the conventional left-to-right, top-to-bottom scan order. Video packets shall also be transmitted following the conventional left-to-right, top-to-bottom macroblock scan order. The last MB of one video packet is guaranteed to immediately precede the first MB of the following video packet in the MB scan order.

load_backward_shape: This is a one-bit flag which when set to ‘1’ implies that the backward shape of the previous VOP in the same layer is copied to the forward shape for the current VOP and the backward shape of the current VOP is decoded from the bitstream. When this flag is set to ‘0’, the forward shape of the previous VOP is copied to the forward_shape of the current VOP and the backward shape of the previous VOP in the same layer is copied to the backward shape of the current VOP. This flag shall be ‘1’ when (1) background_composition is ‘1’ and vop_coded of the previous VOP in the same layer is ‘0’ or (2) background_composition is ‘1’ and the current VOP is the first VOP in the current layer.

If hierarchy_type is “0” and video_object_layer_shape is “binary” (object based spatial scalability), then this flag shall be set to “0”.

--`,,`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

backward_shape_width: This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the backward shape. A zero value is forbidden.

backward_shape_height: This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the backward shape. A zero value is forbidden.

backward_shape_horizontal_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle that includes the backward shape. This is used for decoding and for picture composition.

backward_shape_vertical_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle that includes the backward shape. This is used for decoding and for picture composition.

backward_shape(): The decoding process of the backward shape is identical to the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = “10”).

load_forward_shape: This is a one-bit flag which when set to ‘1’ implies that the forward shape is decoded from the bitstream. This flag shall be ‘1’ when (1) background_composition is ‘1’ and vop_coded of the previous VOP in the same layer is ‘0’ or (2) background_composition is ‘1’ and the current VOP is the first VOP in the current layer.

forward_shape_width: This is a 13-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the forward shape. A zero value is forbidden.

forward_shape_height: This is a 13-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the forward shape. A zero value is forbidden.

forward_shape_horizontal_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle that includes the forward shape. This is used for decoding and for picture composition.

forward_shape_vertical_mc_spatial_ref: This is a 13-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle that includes the forward shape. This is used for decoding and for picture composition.

forward_shape(): The decoding process of the backward shape is identical to the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = “10”).

ref_select_code: This is a 2-bit unsigned integer which specifies prediction reference choices for P- and B-VOPs in enhancement layer with respect to decoded reference layer identified by ref_layer_id. The meaning of allowed values is specified in Table 7-17 and Table 7.18.

resync_marker: This is a binary string of at least 16 zero’s followed by a one ‘0 0000 0000 0000 0001’. For an I-VOP or a VOP where video_object_layer_shape has the value “binary_only”, the resync marker is 16 zeros followed by a one. The length of this resync marker is dependent on the value of vop_fcode_forward, for a P-VOP or a S(GMC)-VOP, and the larger value of either vop_fcode_forward and vop_fcode_backward for a B-VOP. For a P-VOP and a S(GMC)-VOP, the resync_marker is (15+fcode) zeros followed by a one; for a B-VOP, the resync_marker is max(15+fcode,17) zeros followed by a one. It is only present when resync_marker_disable flag is set to ‘0’. A resync marker shall only be located immediately before a macroblock and aligned with a byte.

macroblock_number: This is a variable length code with length between 1 and 14 bits. It identifies the macroblock number within a VOP. The number of the top-left macroblock in a VOP shall be zero. The macroblock number increases from left to right and from top to bottom. The actual length of the code depends on the total number of macroblocks in the VOP calculated according to Table 6-27, the code itself is a binary representation of the macroblock number. If reduced_resolution_vop_enable is equal to one, the length of macroblock_number code shall be determined by as if the total number of macroblocks in a VOP in Table 6-27 be equal to ((video_object_layer_width+15)/16) * ((video_object_layer_height+15)/16).

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-27 — Length of macroblock_number code

length of macroblock_number code	((vop_width+15)/16) * ((vop_height+15)/16)
1	1-2
2	3-4
3	5-8
4	9-16
5	17-32
6	33-64
7	65-128
8	129-256
9	257-512
10	513-1024
11	1025-2048
12	2049-4096
13	4097-8192
14	8193-16384

quant_scale: This is an unsigned integer which specifies the absolute value of quantiser scale to be used for inverse quantising the macroblock of the video packet until updated by any subsequent dquant. The length of this field is specified by the value of the parameter quant_precision. The default length is 5-bits.

header_extension_code: This is a 1-bit flag which when set to '1' indicates the presence of additional fields in the header. When header_extension_code is set to '1', modulo_time_base, vop_time_increment and vop_coding_type are also included in the video packet header. If video_object_layer_shape is not "rectangular", VOP header fields used for the shape decoding (vop_width, vop_height, vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref, change_conv_ratio_disable and vop_shape_coding_type) are also present. If video_object_layer_shape is not "binary only", intra_dc_vlc_thr is also present**.** Furthermore, if the vop_coding_type is equal to either a P, S or B VOP, the appropriate fcodes are also present. Additionally, if the current VOP is an S(GMC)-VOP, sprite_trajectory() is included. And if reduced_resolution_vop_enable is equal to one, vop_reduced_resolution is also present.

use_intra_dc_vlc: The value of this internal flag is set to 1 when the values of intra_dc_thr and the DCT quantiser indicate the usage of the intra DC VLCs as shown in Table 6-25. Otherwise, the value of this flag is set to 0.

motion_marker: This is a 17-bit binary string '1 1111 0000 0000 0001'. It is only present when the data_partitioned flag is set to '1'.In the data partitioning mode, a motion_marker is inserted after the motion data (prior to the texture data). The motion_marker is unique from the motion data and enables the decoder to determine when all the motion information has been received correctly.

dc_marker: This is a 19 bit binary string '110 1011 0000 0000 0001'. It is present when the data_partitioned flag is set to '1'. It is used for I-VOPs. In the data partitioning mode, a dc_marker is inserted into the bitstream after the mcbpc, dquant and DC data but before the ac_pred flag and remaining texture information.

vop_id: This indicates the ID of VOP which is incremented by 1 whenever a VOP is encoded. All '0' value is skipped in vop_id in order to prevent the resync_marker emulation. The bit length of vop_id is the smaller value between (the length of vop_time_increment) + 3 and 15.

vop_id_for_prediction_indication: This is a one-bit flag which indicates the existence of the following vop_id_for_prediction field.

0: vop_id_for_prediction does not exist.
1: vop_id_for_prediction does exist.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

vop_id_for_prediction: This indicates the vop_id of the VOP which is used as the reference VOP of the decoding of the current VOP or Video Packet. When this field exists, the reference VOP shall be replaced with the indicated one. This VOP which is used for prediction may be changed according to the backward channel message. If this field does not exist, the previous VOP is used for prediction, or all MBs in the VOP or Video Packet are coded to Intra MB.

6.3.5.1 Definition of DCECS variable values

The semantic of all complexity estimation parameters is defined at the VO syntax level. DCECS variables represent % values. The actual % values have been converted to 8 bit words by normalization to 256. To each 8 bit word a binary 1 is added to prevent start code emulation (i.e 0% = '00000001', 99.5% = '11111111' and is conventionally considered equal to one). The binary '00000000' string is a forbidden value. The only parameter expressed in their absolute value is the dcecs_vlc_bits parameter expressed as a 4 bit word.

dcecs_opaque: 8 bit number representing the % of luminance and chrominance blocks using opaque coding mode on the total number of blocks (bounding rectangle).

dcecs_transparent: 8 bit number representing the % of luminance and chrominance blocks using transparent coding mode on the total number of blocks (bounding rectangle).

dcecs_intra_cae: 8 bit number representing the % of luminance and chrominance blocks using IntraCAE coding mode on the total number of blocks (bounding rectangle).

dcecs_inter_cae: 8 bit number representing the % of luminance and chrominance blocks using InterCAE coding mode on the total number of blocks (bounding rectangle).

dcecs_no_update: 8 bit number representing the % of luminance and chrominance blocks using no update coding mode on the total number of blocks (bounding rectangle).

dcecs_upsampling: 8 bit number representing the % of luminance and chrominance blocks which need upsampling from 4-4- to 8-8 block dimensions on the total number of blocks (bounding rectangle).

dcecs_intra_blocks: 8 bit number representing the % of luminance and chrominance Intra or Intra+Q coded blocks on the total number of blocks (bounding rectangle).

dcecs_not_coded_blocks: 8 bit number representing the % of luminance and chrominance Non Coded blocks on the total number of blocks (bounding rectangle).

dcecs_dct_coefs: 8 bit number representing the % of the number of DCT coefficients on the maximum number of coefficients (coded blocks).

dcecs_dct_lines: 8 bit number representing the % of the number of DCT8x1 on the maximum number of DCT8x1 (coded blocks).

dcecs_vlc_symbols: 8 bit number representing the average number of VLC symbols for macroblock.

dcecs_vlc_bits: 4 bit number representing the average number of bits for each symbol.

dcecs_inter_blocks: 8 bit number representing the % of luminance and chrominance Inter and Inter+Q coded blocks on the total number of blocks (bounding rectangle).

dcecs_inter4v_blocks: 8 bit number representing the % of luminance and chrominance Inter4V coded blocks on the total number of blocks (bounding rectangle).

dcecs_apm (Advanced Prediction Mode): 8 bit number representing the % of the number of luminance block predicted using APM on the total number of blocks for VOP (bounding rectangle).

dcecs_npm (Normal Prediction Mode): 8 bit number representing the % of luminance and chrominance blocks predicted using NPM on the total number of luminance and chrominance blocks for VOP (bounding rectangle).

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

dcecs_forw_back_mc_q: 8 bit number representing the % of luminance and chrominance predicted blocks on the total number of blocks for VOP (bounding rectangle).

dcecs_halfpel2: 8 bit number representing the % of luminance and chrominance blocks predicted by a half-pel vector on one dimension (horizontal or vertical) on the total number of blocks (bounding rectangle).

dcecs_halfpel4: 8 bit number representing the % of luminance and chrominance blocks predicted by a half-pel vector on two dimensions (horizontal and vertical) on the total number of blocks (bounding rectangle).

dcecs_interpolate_mc_q: 8 bit number representing the % of luminance and chrominance interpolated blocks in % of the total number of blocks for VOP (bounding rectangle).

dcecs_sadct: 8 bit number representing the % of luminance and chrominance blocks coded using SADCT mode in % of the total number of blocks for VOP (bounding rectangle).

dcecs_quarterpel: 8 bit number representing the % of luminance and chrominance blocks predicted using quarter-pel vectors in % of the total number of blocks for VOP (bounding rectangle).

6.3.5.2 Video Plane with Short Header

video_plane_with_short_header() – This data structure contains a video plane using an abbreviated header format. Certain values of parameters shall have pre-defined and fixed values for any video_plane_with_short_header, due to the limited capability of signaling information in the short header format. These parameters having fixed values are shown in Table 6-28.

Table 6-28 — Fixed Settings for video_plane_with_short_header()

Parameter	Value
video_object_layer_shape	“rectangular”
obmc_disable	1
quant_type	0
resync_marker_disable	1
data_partitioned	0
block_count	6
reversible_vlc	0
vop_rounding_type	0
vop_fcode_forward	1
vop_coded	1
interlaced	0
complexity_estimation_disable	1
use_intra_dc_vlc	0
scalability	0
not_8_bit	0
bits_per_pixel	8
colour_primaries	1
transfer_characteristics	1
matrix_coefficients	6

short_video_start_marker: This is a 22-bit start marker containing the value ‘0000 0000 0000 0000 1000 00’. It is used to mark the location of a video plane having the short header format. short_video_start_marker shall be byte aligned by the insertion of zero to seven zero-valued bits as necessary to achieve byte alignment prior to short_video_start_marker.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

temporal_reference: This is an 8-bit number which can have 256 possible values. It is formed by incrementing its value in the previously transmitted video_plane_with_short_header() by one plus the number of non-transmitted pictures (at 30000/1001 Hz) since the previously transmitted picture. The arithmetic is performed with only the eight LSBs.

split_screen_indicator: This is a boolean signal that indicates that the upper and lower half of the decoded picture could be displayed side by side. This bit has no direct effect on the encoding or decoding of the video plane.

document_camera_indicator: This is a boolean signal that indicates that the video content of the VOP is sourced as a representation from a document camera or graphic representation, as opposed to a view of natural video content. This bit has no direct effect on the encoding or decoding of the video plane.

full_picture_freeze_release: This is a boolean signal that indicates that resumption of display updates should be activated if the display of the video content has been frozen due to errors, packet losses, or for some other reason such as the receipt of a external signal. This bit has no direct effect on the encoding or decoding of the video plane.

source_format: This is an indication of the width and height of the rectangular video plane represented by the video_plane_with_short_header. The meaning of this field is shown in Table 6-29. Each of these source formats has the same VOP time increment resolution which is equal to 30000/1001 (approximately 29.97) Hz and the same width:height pixel aspect ratio (288/3):(352/4), which equals 12:11 in relatively prime numbers and which defines a CIF picture as having a width:height picture aspect ratio of 4:3.

Table 6-29 — Parameters Defined by source_format Field

source_format value	Source Format Meaning	vop_width	vop_height	num_macroblocks_in_gob	num_gobs_in_vop
000	reserved	reserved	reserved	reserved	reserved
001	sub-QCIF	128	96	8	6
010	QCIF	176	144	11	9
011	CIF	352	288	22	18
100	4CIF	704	576	88	18
101	16CIF	1408	1152	352	18
110	reserved	reserved	reserved	reserved	reserved
111	reserved	reserved	reserved	reserved	reserved

picture_coding_type: This bit indicates the vop_coding_type. When equal to zero, the vop_coding_type is “I”, and when equal to one, the vop_coding_type is “P”.

four_reserved_zero_bits: This is a four-bit field containing bits which are reserved for future use and equal to zero.

pei: This is a single bit which, when equal to one, indicates the presence of a byte of psupp data following the pei bit.

psupp: This is an eight bit field which is present when pei is equal to one. The pei + psupp mechanism provides for a reserved method of later allowing the definition of backward-compatible data to be added to the bitstream. Decoders shall accept and discard psupp when pei is equal to one, with no effect on the decoding of the video data. The pei and psupp combination pair may be repeated if present. The ability for an encoder to add pei and psupp to the bitstream is reserved for future use.

gob_number: This is a five-bit number which indicates the location of video data within the video plane. A group of blocks (or GOB) contains a number of macroblocks in raster scanning order within the picture. For a given gob_number, the GOB contains the num_macroblocks_per_gob macroblocks starting with macroblock_number = gob_number * num_macroblocks_per_gob. The gob_number can either be read from the bitstream or inferred from the progress of macroblock decoding as shown in the syntax description pseudo-code.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

num_gobs_in_vop: This is the number of GOBs in the VOP. This parameter is derived from the source_format as shown in Table 6-29.

gob_layer(): This is a layer containing a fixed number of macroblocks in the VOP. Which macroblocks which belong to each gob can be determined by gob_number and num_macroblocks_in_gob.

gob_resync_marker: This is a fixed length code of 17 bits having the value '0000 0000 0000 0000 1' which may optionally be inserted at the beginning of each gob_layer(). Its purpose is to serve as a type of resynchronization marker for error recovery in the bitstream. The gob_resync_marker codes may (and should) be byte aligned by inserting zero to seven zero-valued bits in the bitstream just prior to the gob_resync_marker in order to obtain byte alignment. The gob_resync_marker shall not be present for the first GOB (for which gob_number = 0).

gob_number: This is a five-bit number which indicates which GOB is being processed in the VOP. Its value may either be read following a gob_resync_marker or may be inferred from the progress of macroblock decoding. All GOBs shall appear in the bitstream of each video_plane_with_short_header(), and the GOBs shall appear in a strictly increasing order in the bitstream. In other words, if a gob_number is read from the bitstream after a gob_resync_marker, its value must be the same as the value that would have been inferred in the absence of the gob_resync_marker.

gob_frame_id: This is a two bit field which is intended to help determine whether the data following a gob_resync_marker can be used in cases for which the VOP header of the video_plane_with_short_header() may have been lost. gob_frame_id shall have the same value in every GOB header of a given video_plane_with_short_header(). Moreover, if any field among the split_screen_indicator or document_camera_indicator or full_picture_freeze_release or source_format or picture_coding_type as indicated in the header of a video_plane_with_short_header() is the same as for the previous transmitted picture in the same video object, gob_frame_id shall have the same value as in that previous video_plane_with_short_header(). However, if any of these fields in the header of a certain video_plane_with_short_header() differs from that in the previous transmitted video_plane_with_short_header() of the same video object, the value for gob_frame_id in that picture shall differ from the value in the previous picture.

num_macroblocks_in_gob: This is the number of macroblocks in each group of blocks (GOB) unit. This parameter is derived from the source_format as shown in Table 6-29. The count of stuffing macroblocks is not included in this value.

short_video_end_marker: This is a 22-bit end of sequence marker containing the value '0000 0000 0000 0000 1111 11'. It is used to mark the end of a sequence of video_plane_with_short_header(). short_video_end_marker may (and should) be byte aligned by the insertion of zero to seven zero-valued bits to achieve byte alignment prior to short_video_end_marker.

6.3.5.3 Shape coding

bab_type: This is a variable length code between 1 and 7 bits. It indicates the coding mode used for the bab. There are seven bab_types as depicted in Table 6-30. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received. For I-VOPs, the context-switched VLC table of Table B.27 is used. For P-VOPs, B-VOPs, and S(GMC)-VOPs , the context switched table of Table B.28 is used.

Table 6-30 — List of bab_types and usage

bab_type	Semantic	Used in
0	MVDs==0 && No Update	P,B, and S(GMC)- VOPs
1	MVDs!=0 && No Update	P,B, and S(GMC)-VOPs
2	transparent	All VOP types
3	opaque	All VOP types
4	intraCAE	All VOP types
5	MVDs==0 && interCAE	P,B, and S(GMC)-VOPs
6	MVDs!=0 && interCAE	P,B, and S(GMC)-VOPs

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 0, 2 or 3. Opaque means that all pixels of the bab are part of the object. Transparent means that none of the bab pixels belong to the object. IntraCAE means the intra-mode CAE decoding will be required to reconstruct the pixels of the bab. No_update means that motion compensation is used to copy the bab from the previous VOP's binary alpha map. InterCAE means the motion compensation and inter_mode CAE decoding are used to reconstruct the bab. MVDs refers to the motion vector difference for shape.

mvds_x: This is a VLC code between 1 and 18 bits. It represents the horizontal element of the motion vector difference for the bab. The motion vector difference is in full integer precision. The VLC table is shown is Table B.29.

mvds_y: This is a VLC code between 1 and 18 bits. It represents the vertical element of the motion vector difference for the bab. The motion vector difference is in full integer precision. If mvds_x is '0', then the VLC table of Table B.30, otherwise the VLC table of Table B.29 is used.

conv_ratio: This is VLC code of length 1-2 bits. It specifies the factor used for sub-sampling the 16x16 pixel bab. The decoder must up-sample the decoded bab by this factor. The possible values for this factor are 1, 2 and 4 and the VLC table used is given in Table B.31.

scan_type: This is a 1-bit flag where a value of '0' implies that the bab is in transposed form i.e. the BAB has been transposed prior to coding. The decoder must then transpose the bab back to its original form following decoding. If this flag is '1', then no transposition is performed.

binary_arithmetic_code(): This is a binary arithmetic decoder representing the pixel values of the bab. This code may be generated by intra cae or inter cae depending on the bab_type. Cae decoding relies on the knowledge of intra_prob[] and inter_prob[], probability tables given in Annex B.

enh_bab_type: This is a variable length code between 1 and 3 bits. It indicates the bab coding mode used in binary shape enhancement layer coding. There are four enh_bab_types as depicted in Table 6-31. The VLC tables used depend on the decoding context i.e. the bab_types of blocks in lower reference layer.

Table 6-31 — List of enh_bab_types and usage

enh_bab_type	Semantic	Used in
0	intra NOT_CODED	P-,B-VOPs
1	intra CODED	P-,B-VOPs
2	inter NOT_CODED	B- VOPs
3	inter CODED	B- VOPs

The enh_bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the enh_bab_type = 0 or 2. NOT_CODED means that motion compensation is used to copy the bab from the reference bab's binary alpha map. In intra NOT_CODED mode, the upsampled bab from the collocated block in lower reference layer is used for the reference bab. In inter NOT_CODED mode, motion compensated bab in the previous VOP of the current layer is used for the reference bab. And the motion vector of the collocated block in the lower reference layer is used for the motion compensation. Each component of the lower layer shape motion vectors are scaled by the up-sampling ratio according to subclause 7.5.4.5 for referencing in the enhancement layer. Intra CODED means Scan Interleaving(SI) based CAE decoding will be required to reconstruct the pixels of the current bab. Inter CODED means the motion compensation and inter_mode CAE decoding are used to reconstruct the current bab.

enh_binary_arithmetic_code() – This is a binary arithmetic decoder representing the pixel values of the bab and bab type of Scan Interleaving (SI) method. This code may be generated by SI decoding method or inter cae depending on the enh_bab_type. When enh_bab_type==1, the first decoded value represents the bab type of SI decoding method ("0": transitional bab, "1": exceptional bab). And the other decoded values represent the pixel values of the bab. If the bab type of SI is "transitional bab", only transitional pixels in the coded-scan-lines are decoded. Otherwise, for "exceptional bab", all of the pixels in the coded-scan-lines are decoded. This binary value decoding relies on the knowledge of enh_bab_type_prob[], and enh_intra_v_prob[] and

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

enh_intra_h_prob[]probability tables given in Annex B. When enh_bab_type==3, this binary arithmetic decoder represents the pixel values of the bab. This code is generated by inter cae. This binary value decoding relies on the knowledge of inter_prob[], probability tables given in Annex B.

6.3.5.4 Sprite coding

warping_mv_code(dmv : The codeword for each differential motion vector consists of a VLC indicating the length of the dmv code (dmv_length) and a FLC, dmv_code-, with dmv_length bits. The codewords are listed in Table B.33.

brightness_change_factor (): The codeword for brightness_change_factor consists of a variable length code denoting brightness_change_factor_size and a fix length code, brightness_change_factor, of brightness_change_factor_size bits (sign bit included). The codewords are listed in Table B.34.

send_mb():This function returns 1 if the current macroblock has already been transmitted. Otherwise it returns 0.

piece_quant: This is a 5-bit unsigned interger which indicates the quantiser scale to be used for a sprite-piece until updated by a subsequent dquant. The piece_quant carries the binary representation of quantiser scale values from 1 to 31 in steps of 1.

piece_width: This value specifies the width of the sprite piece measured in macroblock units.

piece_height: This value specifies the height of the sprite piece measured in macroblock units.

piece_xoffset: This value specifies the horizontal offset location, measured in macroblock units from the left edge of the sprite object, for the placement of the sprite piece into the sprite object buffer at the decoder.

piece_yoffset: This value specifies the vertical offset location, measured in macroblock units from the top edge of the sprite object.

decode_sprite_piece (): It decodes a selected region of the sprite object or its update. It also decodes the parameters required by the decoder to properly incorporate the pieces. All the static-sprite-object pieces will be encoded using a subset of the I-VOP syntax. And the static-sprite-update pieces use a subset of the P-VOP syntax. The sprite update is defined as the difference between the original sprite texture and the reconstructed sprite assembled from all the sprite object pieces.

sprite_shape_texture(): For the static-sprite-object pieces, shape and texture are coded using the macroblock layer structure in I-VOPs. And the static-sprite-update pieces use the P-VOP inter-macroblock syntax -- except that there are no motion vectors and shape information included in this syntax structure. Macroblocks raster scanning is employed to encode a sprite piece; however, whenever the scan encounters a macroblock which has been part of some previously sent sprite piece, then the block is not coded and the corresponding macroblock layer is empty.

6.3.6 Macroblock related

not_coded: This is a 1-bit flag which signals if a macroblock is coded or not. When set to'1' it indicates that a macroblock is not coded and no further data is included in the bitstream for this macroblock (with the exception of alpha data that may be present). The decoder shall treat this macroblock as 'inter' with motion vector equal to zero and no DCT coefficient data for P-VOPs, and the decoder shall treat this macroblock as 'GMC macroblock (i.e. prediction using the global motion compensated image)' with no motion vector data and no DCT coefficient data for S(GMC)-VOPs. When set to '0' it indicates that the macroblock is coded and its data is included in the bitstream.

mcbpc: This is a variable length code that is used to derive the macroblock type and the coded block pattern for chrominance. It is always included for coded macroblocks. Table B.6 and Table B.7 list all allowed codes for mcbpc in I-, P-, and S(GMC)- VOPs respectively. The values of the column "MB type" in these tables are used as the variable "derived_mb_type" which is used in the respective syntax part for motion and texture decoding. In P-VOPs using the short video header format (i.e., when short_video_header is 1), mcbpc codes indicating macroblock type 2 shall not be used.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

mcsel: This is a 1-bit flag that specifies the reference image of each macroblock in S-VOPs. This flag is present only when sprite_enable == “GMC,” vop_coding_type == “S”, and the macroblock type specified by mcbpc is “inter” or “inter+q”. mcsel indicates whether the global motion compensated image or the previous reconstructed VOP is referred to for interframe prediction. This flag is set to ‘1’ when GMC is used for the macroblock, and is set to ‘0’ if local MC is used. If mcsel = “1”, local motion vectors are not transmitted. The default value for mcsel is ‘1’ (i.e. prediction using GMC) when sprite_enable == “GMC,” vop_coding_type == “S”, and not_coded == ‘1’.

ac_pred_flag: This is a 1-bit flag which when set to ‘1’ indicates that either the first row or the first column of AC coefficients are differentially coded for intra coded macroblocks.

cbpy: This variable length code represents a pattern of non-transparent luminance blocks with at least one non intra DC transform coefficient, in a macroblock. Table B.8 to Table B.11 indicate the codes and the corresponding patterns they indicate for the respective cases of intra- and inter-MBs.

dquant: This is a 2-bit code which specifies the change in the quantiser scale, for I-, P-, and S(GMC)-VOPs. Table 6-32 lists the codes and the differential values they represent. The value of the quantiser scale lies in range of 1 to $2^{quant_precision}$-1; if the value of quantiser scale after adding dquant value is less than 1 or exceeds $2^{quant_precision}$-1, it shall be correspondingly clipped to 1 and $2^{quant_precision}$-1. If quant_precision takes its default value of 5, the range of allowed values for the quantiser scale is [1:31].

Table 6-32 — dquant codes and corresponding values

dquant code	value
00	-1
01	-2
10	1
11	2

co_located_not_coded: The value of this internal flag is set to 1 when the current VOP is a B-VOP, the future reference VOP is a P-VOP, and the co-located macroblock in the future reference VOP is skipped (i.e. coded as not_coded = '1'). Otherwise the value of this flag is set to 0. The co-located macroblock is the macroblock which has the same horizontal and vertical index with the current macroblock in the B-VOP. If the co-located macroblock lies outside of the bounding rectangle, this macroblock is considered to be not skipped.

modb: This is a variable length code present only in coded macroblocks of B-VOPs. It indicates whether mb_type and/or cbpb information is present for a macroblock. The codes for modb are listed in Table B.3.

mb_type: This variable length code is present only in coded macroblocks of B-VOPs. Further, it is present only in those macroblocks for which one motion vector is included. The codes for mb_type are shown in Table B.4 for B-VOPs for no scalability and in Table B.5 for B-VOPs with scalability. When mb_type is not present (i.e. modb==‘1’) for a macroblock in a B-VOP, the macroblock type is set to the default type. The default macroblock type for the enhancement layer of spatially scalable bitstreams (i.e. ref_select_code == '00' && scalability = '1') is "forward mc + Q". Otherwise, the default macroblock type is "direct".

cbpb: This is a 3 to 6 bit code representing coded block pattern in B-VOPs, if indicated by modb. Each bit in the code represents a coded/no coded status of a block; the leftmost bit corresponds to the top left block in the macroblock. For each non-transparent block with coefficients, the corresponding bit in the code is set to ‘1’. In case no coefficients are coded for all the non-transparent blocks in the macroblock, modb shall be set to the value indicating cbpb is not present (i.e. modb==‘1’ or ‘01’) and cbpb shall not be included in the bitstream for this macroblock.

dbquant: This is a variable length code which specifies the change in quantiser scale for B-VOPs. Table 6-33 lists the codes and the differential values they represent. If the value of the quantiser scale after adding dbquant value is less than 1 or exceeds $2^{quant_precision}$-1, it shall be correspondingly clipped to 1 and $2^{quant_precision}$-1. If quant_precision takes its default value of 5, the range of allowed values for the quantzer for B-VOP is [1:31].

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-33 — dbquant codes and corresponding values

dbquant code	value
10	-2
0	0
11	2

coda_i: This is a one-bit flag which is set to "1" to indicate that all the values in the grayscale alpha macroblock are equal to 255 (AlphaOpaqueValue). When set to "0", this flag indicates that one or more 8x8 blocks are coded according to cbpa.

ac_pred_flag_alpha: This is a one-bit flag which when set to '1' indicates that either the first row or the first column of AC coefficients are to be differentially decoded for intra alpha macroblocks. It has the same effect for alpha as the corresponding luminance flag.

cbpa: This is the coded block pattern for grayscale alpha texture data. For I, P, S(GMC), and B VOPs, this VLC is exactly the same as the inter (P or S(GMC)) cbpy VLC described in Table B.8 – Table B.11. cbpa is followed by the alpha block data which is coded in the same way as texture block data. Note that grayscale alpha blocks with alpha all equal to zero (transparent) are not included in the bitstream.

coda_pb: This is a VLC indicating the coding status for P, S(GMC), or B alpha macroblocks. The semantics are given in the table below (Table 6-34). When this VLC indicates that the alpha macroblock is all opaque, this means that all values are set to 255 (AlphaOpaqueValue).

Table 6-34 — coda_pb codes and corresponding values

coda_pb	Meaning
1	alpha residue all zero
01	alpha macroblock all opaque
00	alpha residue coded

6.3.6.1 MB Binary Shape Coding

bab_type: This defines the coding type of the current bab according to Table B.27 and Table B.28 for intra and inter mode, respectively.

mvds_x: This defines the size of the x-component of the differential motion vector for the current bab according to Table B.29.

mvds_y: This defines the size of the y-component of the differential motion vector for the current bab according to Table B-29 if mvds_x!=0 and according to Table B.30 if mvds_x==0.

conv_ratio: This defines the upsampling factor according to Table B.31 to be applied after decoding the current shape information.

scan_type: This defines according to Table 6-35 whether the current bordered to be decoded bab and the eventual bordered motion compensated bab need to be transposed.

Table 6-35 — scan_type

scan_type	meaning
0	transpose bab as in matrix transpose
1	do not transpose

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

binary_arithmetic_code(): This is a binary arithmetic decoder that defines the context dependent arithmetically to be decoded binary shape information. The meaning of the bits is defined by the arithmetic decoder according to subclause 7.5.3

enh_bab_type: This defines the coding type of the current bab in the enhancement layer according to Table 7.7 and Table 7.8 for P-VOP and B-VOP coding, respectively.

enh_binary_arithmetic_code(): This is a binary arithmetic decoder that defines the context dependent arithmetically to be decoded binary shape information in the enhancement layer. The meaning of the bits is defined by the arithmetic decoder according to subclause 7.5.3.

6.3.6.2 Motion vector

horizontal_mv_data: This is a variable length code, as defined in Table B.12, which is used in motion vector decoding as described in subclause 7.6.3.

vertical_mv_data: This is a variable length code, as defined in Table B.12, which is used in motion vector decoding as described in subclause 7.6.3.

horizontal_mv_residual: This is an unsigned integer which is used in motion vector decoding as described in subclause 7.6.3. The number of bits in the bitstream for horizontal_mv_residual, r_size, is derived from either vop_fcode_forward or vop_fcode_backward as follows;

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

vertical_mv_residual: This is an unsigned integer which is used in motion vector decoding as described in subclause 7.6.3. The number of bits in the bitstream for vertical_mv_residual, r_size, is derived from either vop_fcode_forward or vop_fcode_backward as follows;

r_size = vop_fcode_forward - 1 or r_size = vop_fcode_backward - 1

6.3.6.3 Interlaced Information

dct_type: This is a 1-bit flag indicating whether the macroblock is frame DCT coded or field DCT coded. If this flag is set to “1”, the macroblock is field DCT coded; otherwise, the macroblock is frame DCT coded. This flag is only present in the bitstream if the interlaced flag is set to “1” and the macroblock is coded (coded blcok pattern is non-zero) or intra-coded. Boundary blocks are always coded in frame-based mode.

field_prediction: This is a 1-bit flag indicating whether the macroblock is field predicted or frame predicted. This flag is set to ‘1’ when the macroblock is predicted using field motion vectors. If it is set to ‘0’ then frame prediction (16x16 or 8x8) will be used. This flag is only present when interlaced == ‘1’ for the following types of macrloblocks: a macroblock in a P-VOP with derived_mb_type < 2; a non-direct mode macrloblock in a B-VOP; or a macroblocks in an S (GMC)-VOP with mcsel == ‘0’.

forward_top_field_reference: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the top field. When this flag is set to ‘0’, the top field is used as the reference field. If it is set to ‘1’ then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to “1” and the macroblock is not backward predicted.

forward_bottom_field_reference: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the bottom field. When this flag is set to ‘0’, the top field is used as the reference field. If it is set to ‘1’ then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to “1” and the macroblock is not backward predicted.

backward_top_field_reference: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the top field. When this flag is set to ‘0’, the top field is used as the reference field. If it is set to ‘1’ then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to “1” and the macroblock is not forward predicted.

--`,,`-`-`,,`,,`,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

backward_bottom_field_reference: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the bottom field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the field_prediction flag is set to "1" and the macroblock is not forward predicted.

6.3.7 Block related

intra_dc_coefficient: This is a fixed length code that defines the value of an intra DC coefficient when the short video header format is in use (i.e., when short_video_header is "1"). It is transmitted as a fixed length unsigned integer code of size 8 bits, unless this integer has the value 255. The values 0 and 128 shall not be used – they are reserved. If the integer value is 255, this is interpreted as a signalled value of 128. The integer value is then multiplied by a dc_scaler value of 8 to produce the reconstructed intra DC coefficient value.

dct_dc_size_luminance: This is a variable length code as defined in Table B.13 that is used to derive the value of the differential DC coefficients of luminance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

dct_dc_differential: This is a variable length code as defined in Table B.15 that is used to derive the value of the differential DC coefficients in blocks in intra macroblocks. After identifying the category of the DC coefficient in size from dct_dc_size_luminance or dct_dc_size_chrominance, this value denotes which actual difference in that category occurred.

dct_dc_size_chrominance: This is a variable length code as defined in Table B.14 that is used to derive the value of the differential DC coefficients of chrominance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

pattern_code[i]: The value of this internal flag is set to 1 if the block or alpha block with the index value i includes one or more DCT coefficients that are decoded using at least one of Table B.16 to Table B.25. Otherwise the value of this flag is set to 0.

6.3.7.1 Alpha block related

dct_dc_size_alpha: This is a variable length code for coding the alpha block DC coefficient. Its semantics are the same as dct_dc_size_luminance in subclause 6.3.7.

6.3.8 Still texture object

still_texture_object_start_code: The still_texture_object_start_code is a string of 32 bits. The first 24 bits are '0000 0000 0000 0000 0000 0001' and the last 8 bits are defined in Table 6-3.

texture_tile_start_code: The texture_tile_start_code is a string of 32 bits. The 32 bits are '0000 0000 0000 0000 0000 0001 1100 0001' in binary. The texture_tile_start code marks the start of a new tile.

tiling_disable: This field indicates the succeeding bitstream contains a structure of tile when the field is '0'.

tile_width: This is a 15-bit unsigned integer which specifies horizontal size, in pexel unit, of the rectangle. When texture_object_layer_shape=='00', this value must be lower than texture_object_layer_width and a zero value is forbidden. When texture_object_layer_width is not a multilple of tile_width, the horizontal size of tile in the last column is derived by texture_object_layer_width%tile_width. When texture_object_layer_shape=='01', object_width is used instead of texture_object_layer_width. The value of tile_width shall be divisible by two.

tile_height: This is a 15-bit unsigned integer which specifies vertical size, in pexel unit, of the rectangle. When texture_object_layer_shape=='00', this value must be lower than texture_object_layer_height and a zero value is forbidden. When texture_object_layer_height is not a multilple of tile_height, the vertical size of tile in the last row is derived by texture_object_layer_height%tile_height. When texture_object_layer_shape=='01', object_height is used instead of texture_object_layer_height. The value of tile_height shall be divisible by two.

number_of_tiles: This is a 16-bit of unsigned integer specifying the number of tiles encoded in this bitstream. When texture_object_layer_shape=='00', the value is derived from CEIL(texture_object_layer_width÷tile_width) *

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CEIL(texture_object_layer_height÷tile_height), where CEIL() rounds up to the nearest integer. When texture_object_layer_shape==’01’, the value is derived from CEIL(object_width÷tile_width) * CEIL(object_height÷tile_height).

tiling_jump_table_enable: This field indicates the succeeding bitstream contains a size of bitstream for each tile when the field is ‘1’.

tile_size_high: This is a left part of 16-bit of a unsigned integer in 32-bit which indicates a size of bitstream containing the corresponding tile in byte unit.

tile_size_low: This is a right part of 16-bit of a unsigned integer in 32-bit which indicates a size of bitstream containing the corresponding tile in byte unit. The real size of bitstream for a certain tile is derived from ‘tile_size_high<<16+tile_size_low’.

tile_id: This is given by 16-bits representing one of the values in the range of ‘0000’ to ‘FFFF’ in hexadecimal starting from top-left ended to bottom-right. The field uniquely identifies each tile.

texture_error_resilience_disable: This is a one-bit flag which when set to ‘0‘ indicates that the Still Texture Object is operating in error resilience mode.

target_segment_length: This parameter specifies the minimum number of bits in a segment within a packet before adding a segment marker.

decode_segment_marker(): This function will decode the arithmetically encoded segment marker. The arithmetic model used is the initial model used in decoding type information for the color in which the segment marker is. When in error free case, a ZTR symbol will be decoded as the segment marker.

texture_object_id: This is given by 16-bits representing one of the values in the range of ‘0000 0000 0000 0000’ to ‘1111 1111 1111 1111’ in binary. The texture_object_id uniquely identifies a texture object layer.

wavelet_filter_type: This field indicates the arithmetic precision which is used for the wavelet decomposition as the following:

Table 6-36 — Wavelet type

wavelet_filter_type	Meaning
0	integer
1	Double float

wavelet_download: This field indicates if the 2-band filter bank is specificed in the bitstream:

Table 6-37 — Wavelet downloading flag

wavelet_download	meaning
0	default filters
1	specified in bitstream

The default filter banks are described in subclause B.2.2.

wavelet_decomposition_levels: This field indicates the number of levels in the wavelet decomposition of the texture.

scan_direction: This field indicates the scan order of AC coefficients. In single-quant and multi-quant mode, if this flag is `0’, then the coefficients are scanned in the tree-depth fashion. If it is `1’, then they are scanned in the subband by subband fashion. In bilevel_quant mode, if the flag is `0’, then they are scanned in bitplane by bitplane

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fashion. Within each bitplane, they are scanned in a subband by subband fashion. If it is “1”, they are scanned from the low wavelet decomposition layer to high wavelet decomposition layer. Within each wavelet decomposition layer, they are scanned from most significant bitplane down to the least significant bitplane.

start_code_enable: If this flag is enabled (disable =0; enabled = 1), the start code followed by an ID to be inserted in to each spatial scalability layer and/or each SNR scalability layer.

texture_object_layer_shape: This is a 2-bit integer defined in Table 6-38. It identifies the shape type of a texture object layer.

Table 6-38 — Texture Object Layer Shape type

texture_object_layer_shape	Meaning
00	rectangular
01	binary
10	reserved
11	grayscale

quantisation_type: This field indicates the type of quantisation as shown in Table 6-39.

Table 6-39 — The quantisation type

quantisation_type	Code
single quantiser	01
multi quantiser	10
bi-level quantiser	11

spatial_scalability_levels: This field indicates the number of spatial scalability layers supported in the bitstream. This number can be from 1 to wavelet_decomposition_levels.

use_default_spatial_scalability: This field indicates how the spatial scalability levels are formed. If its value is one, then default spatial scalability is used, starting from (¼)^(spatial_scalability_levels-1)-th of the full resolution up to the full resolution, where ^ is a power operation. If its value is zero, the spatial scalability is specified by wavelet_layer_index described below.

wavelet_layer_index: This field indicates the identification number of wavelet_decomposition layer used for spatial scalability. The index starts with 0 (i.e., root_band) and ends at (wavelet_decomposition_levels–1) (i.e., full resolution).

uniform_wavelet_filter: If this field is “1”, then the same wavelet filter is applied for all wavelet layers. If this field is “0”, then different wavelet filters may be applied for the wavelet decomposition. Note that the same filters are used for both luminance and chromanence. Since the chromanence’s width and height is half that of the luminance, the last wavelet filter applied to the luminance is skipped when the chromanence is synthesized.

wavelet_stuffing: These 3 stuffing bits are reserved for future expansion. It is currently defined to be ‘111’.

texture_object_layer_width: The texture_object_layer_width is a 15-bit unsigned integer representing the width of the displayable part of the luminance component in pixel units. A zero value is forbidden.

texture_object_layer_height: The texture_object_layer_height is a 15-bit unsigned integer representing the height of the displayable part of the luminance component in pixel units. A zero value is forbidden.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

horizontal_ref: This is a 15-bit integer which specifies, in pixel units, the horizontal position of the top left of the rectangle defined by horizontal size of object_width. The value of horizontal_ref shall be divisible by two. This is used for decoding and for picture composition.

vertical_ref: This is a 15-bit integer which specifies, in pixel units, the vertical position of the top left of the rectangle defined by vertical size of object_height. The value of vertical_ref shall be divisible by two. This is used for decoding and for picture composition.

object_width: This is a 15-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the object. A zero value is forbidden.

object_height: This is a 15-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the object. A zero value is forbidden.

quant_byte: This field defines one byte of the quantisation step size for each scalability layer. A zero value is forbidden. The quantisation step size parameter, quant, is decoded using the function get_param( ): quant = get_param(7).

max_bitplanes: This field indicates the number of maximum bitplanes in all three quantisation modes.

texture_tile_type: This is a 2-bit integer defined in Table 6-40. It identifies the shape type of a texture object in a tile when texture_object_layer_shape is “01”.

Table 6-40 — Texture Tile Shape

texture_tile_type	Meaning
00	Forbiden
01	opaque tile
10	Boundary tile
11	Transparent tile

next_texture_marker ():This function performs a similar operation as next_start_code(), but for texture_marker.

texture_marker – This is a binary string of at least 16 zero’s followed by a one ‘0 0000 0000 0000 0001’. It is only present when texture_error_resilience_disable flag is set to ‘0’. A texture marker shall only be located immediately before a texture packet and aligned with a byte.

TU_first – This parameter specifies the number of the first texture unit within the texture packet. This parameter is decoded by the function get_param().

TU_last – This parameter specifies the number of the last texture unit within the texture packet. This parameter is decoded by the function get_param().

header_extention_code – This is a one-bit flag which when set to ‘1‘ indicates that additional header information is sent in the texture packet. This bit must have value 1 in the first packet of a texture object.

target_segment_length – This parameter specifies the minimum number of bits in a segment within a packet before adding a segment marker.

6.3.8.1 Texture Layer Decoding

arith_decode_highbands_td(): This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands (all bands except DC band) within a single tree block. The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described in subclause B.2.2. This decoder uses only integer arithmetic. It also uses an adaptive probability model based on the frequency counts of the previously decoded symbols. The maximum range (or precision) specified is

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(2^16) - 1 (16 bits). The maximum frequency count for the magnitude and residual models is 127, and for all other models it is 127. The arithmetic coder used is identical to the one used in arith_decode_highbands_bilevel_td().

texture_spatial_layer_start_code: The texture_spatial_layer_start_code is a string of 32 bits. The 32 bits are '0000 0000 0000 0000 0000 0001 1011 1111' in binary. The texture_spatial_layer_start_code marks the start of a new spatial layer.

texture_spatial_layer_id: This is given by 5-bits representing one of the values in the range of '00000' to '11111' in binary. The texture_spatial_layer_id uniquely identifies a spatial layer.

arith_decode_highbands_bb(): This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands (all bands except DC band) within a single band. The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described in subclause B.2.2. This decoder uses arithmetic. It also uses an adaptive probability model based on the frequency counts of the previously decoded symbols. The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count for the magnitude and residual models is 127, and for all other models it is 127.

snr_scalability_levels: This field indicates the number of levels of SNR scalability supported in this spatial scalability level.

texture_snr_layer_start_code: The texture_snr_layer_start_code is a string of 32 bits. The 32 bits are '0000 0000 0000 0000 0000 0001 1100 0000' in binary. The texture_snr_layer_start_code marks the start of a new snr layer.

texture_snr_layer_id: This is given by 5-bits representing one of the values in the range of '00000' to '11111' in binary. The texture_snr_layer_id uniquely identifies an SNR layer.

NOTE All the start codes start at the byte boundary. Appropriate number of bits is stuffed before any start code to byte-align the bitstream.

all_nonzero: This flag indicates whether some of the subbands of the current layer contain only zero coefficients. The value '0' for this flag indicates that one or more of the subbands contain only zero coefficients. The value '1' for this flag indicates the all the subbands contain some nonzero coefficients.

all_zero: This flag indicates whether all the coefficients in the current layer are zero or not. The value '0' for this flag indicates that the layer contains some nonzero coefficients. The value '1' for this flag indicates that the layer only contains zero coefficients, and therefore the layer is skipped.

lh_zero, hl_zero, hh_zero: This flag indicates whether the LH/HL/HH subband of the current layer contains only all zero coefficients. The value '1' for this flag indicates that the LH/HL/HH subband contains only zero coefficients, and therefore the subband is skipped. The value '0' for this flag indicates that the LH/HL/HH subband contains some nonzero coefficients

arith_decode_highbands_bilevel_bb(): This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands in the bilevel_quant mode (all bands except DC band). The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described. The arith_decode_highbands_bilevel() function uses bitplane scanning, and a different probability model as described in subclause B.2.2. In this mode, The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count is 127. It uses the lh/hl/hh_zero flags to see if any of the LH/HL/HH are all zero thus not decoded. For example if lh_zero=1 and hh_zero=1 only hl_zero is decoded.

arith_decode_highbands_bilevel_td(): This is an arithmetic decoder for decoding the quantised coefficient values of the higher bands in the bilevel_quant mode (all bands except DC band). The bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of the uniform probability distribution models described. The arith_decode_highbands_bilevel() function uses bitplane scanning, and a different probability model as described in subclause B.2.2. In this mode, The maximum range (or precision) specified is (2^16) - 1 (16 bits). The maximum frequency count is 127. It uses the lh/hl/ll_zero flags to see if any of the LH/HL/HH are all zero thus not decoded. For example if lh_zero=1 and hh_zero=1 only hl_zero is decoded.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

lowpass_filter_length: This field defines the length of the low pass filter in binary ranging from “0001” (length of 1) to “1111” (length of 15.)

highpass_filter_length: This field defines the length of the high pass filter in binary ranging from “0001” (length of 1) to “1111” (length of 15.)

filter_tap_integer: This field defines an integer filter coefficient in a 16 bit signed integer. The filter coefficients are decoded from the left most tap to the right most tap order.

filter_tap_float_high: This field defines the left 16 bits of a floating filter coefficient which is defined in 32-bit IEEE floating format. The filter coefficients are decoded from the left most tap to the right most tap order.

filter_tap_float_low: This field defines the right 16 bits of a floating filter coefficient which is defined in 32-bit IEEE floating format. The filter coefficients are decoded from the left most tap to the right most tap order.

integer_scale: This field defines the scaling factor of the integer wavelet, by which the output of each composition level is divided by an integer division operation. A zero value is forbidden.

mean: This field indicates the mean value of one color component of the texture.

quant_dc_byte: This field indicates the quantisation step size for one color component of the DC subband. A zero value is forbidden. The quantisation step size parameter, quant_dc, is decoded using the function get_param( ): quant_dc = get_param(7); where get_param() function is defined in the description of band_offset_byte.

band_offset_byte: This field defines one byte of the absolute value of the parameter band_offset. This parameter is added to each DC band coefficient obtained by arithmetic decoding. The parameter band_offset is decoded using the function get_param( ):

```
band_offset = -get_param(7);
```

where function get_param() is defined as

```
int get_param(int  nbit)
{
int count = 0;
int word =0;
int value = 0;
int module = 1<<(nbit);

    do{
word= get_next_word_from_bitstream( nbit+1);
value += (word  & (module-1) ) << (count * nbit);
count ++;
} while( word>> nbit);
    return value;
}
```

The function get_next_word_from_bitstream(x) reads the next x bits from the input bitstream.

band_max_byte: This field defines one byte of the maximum value of the DC band. The parameter band_max_value is decoded using function get_param(). The number of maximum bitplanes for DC band is derived from CEIL($\log_2$(band_max_value+1))

```
band_max_value =  get_param(7);
```

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

arith_decode_dc(): This is an arithmetic decoder for decoding the quantised coefficient values of DC band only. No zerotree symbol is decoded since the VAL is assumed for all DC coefficient values. This bitstream is generated by an adaptive arithmetic encoder. The arithmetic decoding relies on the initialization of a uniform probability distribution model described in subclause B.2.2. The arith_decode_dc() function uses the same arithmetic decoder as described in arith_decode_highbands_td() but it uses different scanning, and a different probability model (*DC*).

6.3.8.2 Shape Object decoding

change_conv_ratio_disable: This specifies whether conv_ratio is encoded at the shape object decoding function. If it is set to "1" when disable.

sto_constant_alpha: This is a 1-bit flag when set to '1', the opaque alpha values of the binary mask are replaced with the alpha value specified by sto_constant_alpha_value.

sto_constant_alpha_value: This is an 8-bit code that gives the alpha value to replace the opaque pixels in the binary alpha mask. Value '0' is forbidden.

marker_bit: This is one-bit that shall be set to 1. This bit prevents emulation of start codes.

sto_shape_coded_layers: This is a 4-bit unsigned integer to indicate the number of enhancement layers to contained in the bitstream.

texture_shape_layer_start_code: This is a string of 32 bits. The first 24 bits are '0000 0000 0000 0000 0000 0001' and the last 8 bits '1100 0010' (0xC2). This texture_shape_layer_start_code marks the start of a new shape enhancement layer.

texture_shape_layer_id: This is given by a 5-bit number representing one of the values in the range of '00000' to '11111' in binary. The texture_shape_layer_id uniquely identifies a shape spatial layer.

texture_spatial_layer_start_code: This is a string of 32 bits. The first 24 bits are '0000 0000 0000 0000 0000 0001' and the last 8 bits '1011 1111' (0xBF). This texture_spatial_layer_start_code marks the start of texture decoding.

texture_spatial_layer_id: This is given by a 5-bit number representing one of the values in the range of '00000' to '11111' in binary. This texture_spatial_layer_id uniquely identifies the start of texture decoding.

shape_base_layer_height_blocks(): This is a function that returns the number of shape blocks in vertical directions in the base layer. In the case that tiling_disable is '1' object_height will be used for the number. The number is given by ((object_height>>wavelet_decomposition_levels)+15)/16.

If tiling_disable is '0' tiling_height will be used instead of object_height. The number is given by ((tile_height>>wavelet_decomposition_levels) +15)/16.

shape_base_layer_width_blocks(): This is a function that returns the number of shape blocks in horizontal directions in the base layer. In the case that tiling_disable is '1' object_width will be used for the number. The number is given by ((object_width>>wavelet_decomposition_levels) +15)/16.

If tiling_disable is '0' tiling_width will be used instead of object_width. The number is given by ((tile_width>>wavelet_decomposition_levels) +15)/16.

bab_type: This is a variable length code of 1-2 bits. It indicates the coding mode used for the bab. There are three bab_types as depicted in Table 6-41. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-41 — List of bab_types and usage

bab_type	Semantic	code
2	transparent	10
3	opaque	0
4	intraCAE	11

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 2 or 3. Opaque means that all pixels of the bab are part of the object. Transparent means that none of the bab pixels belong to the object. IntraCAE means the intra-mode CAE decoding will be required to reconstruct the pixels of the bab.

conv_ratio: This is VLC code of length 1-2 bits. It specifies the factor used for sub-sampling the 16x16 pixel bab. The decoder must up-sample the decoded bab by this factor. The possible values for this factor are 1, 2 and 4 and the VLC table used is given in Table B.31.

scan_type: This is a 1-bit flag where a value of '0' implies that the bab is in transposed form i.e. the bab has been transposed prior to coding. The decoder must then transpose the bab back to its original form following decoding. If this flag is '1', then no transposition is performed.

binary_arithmetic_decode(): This is a binary arithmetic decoder representing the pixel values of the bab. Cae decoding relies on the knowledge of intra_prob[], probability tables given in Annex B.

shape_enhanced_layer_height_blocks(): This is a function that returns the number of shape blocks in vertical directions in the enhancement layer. In the case that tiling_disable is '1', object_height will be used for the number. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((object_height>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

If tiling_disable is '0' tiling_height will be used instead of object_height. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((tile_height>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

The value of bab_size (size of the coded bab) is defined in subclause 7.10.6.2.1.

shape_enhanced_layer_width_blocks(): This is a function that returns the number of shape blocks in horizontal directions in the enhancement layer. In the case that tiling_disable is '1', object_width will be used for the number. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((object_width>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

If tiling_disable is '0' tiling_width will be used instead of object_width. If the current coding layer is the *L*-th layer in the bitstream, the number of blocks is given by

((tile_width>>(wavelet_decomposition_levels–*L*–1))+bab_size–1)/bab_size.

enh_binary_arithmetic_decode(): The first decoded value denotes BAB type of Scan Interleaving (SI) method (SI_bab_type: "0": transitional BAB, "1": exceptional BAB). And the other decoded values represent the pixel values of the current BAB. If the BAB is a transitional BAB, only transitional pixels are decoded. Otherwise all of the pixels are decoded. This binary value decoding relies on the knowledge of SI_bab_type_prob[], enh_intra_v_prob[] and enh_intra_h_prob[], sto_SI_bab_type_prob_even[], sto_enh_odd_prob0[], sto_enh_even_prob0[], sto_enh_odd_prob1[], and sto_enh_even_prob1[] probability tables given in Annex B.

--`,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.9 Mesh object

mesh_object_start_code: The mesh_object_start_code is the bit string '000001BC' in hexadecimal. It initiates a mesh object.

6.3.9.1 Mesh object plane

mesh_object_plane_start_code: The mesh_object_plane_start_code is the bit string '000001BD' in hexadecimal. It initiates a mesh object plane.

is_intra: This is a 1-bit flag which when set to '1' indicates that the mesh object is coded in intra mode. When set to '0' it indicates that the mesh object is coded in predictive mode.

6.3.9.2 Mesh geometry

mesh_type_code: This is a 2-bit integer defined in Table 6-42. It indicates the type of initial mesh geometry to be decoded.

Table 6-42 — Mesh type code

mesh type code	mesh geometry
00	forbidden
01	uniform
10	Delaunay
11	reserved

nr_of_mesh_nodes_hor: This is a 10-bit unsigned integer specifying the number of nodes in one row of a uniform mesh.

nr_of_mesh_nodes_vert: This is a 10-bit unsigned integer specifying the number of nodes in one column of a uniform mesh.

mesh_rect_size_hor: This is a 8-bit unsigned integer specifying the width of a rectangle of a uniform mesh (containing two triangles) in half pixel units.

mesh_rect_size_vert: This is a 8-bit unsigned integer specifying the height of a rectangle of a uniform mesh (containing two triangles) in half pixel units.

triangle_split_code: This is a 2-bit integer defined in Table 6-43. It specifies how rectangles of a uniform mesh are split to form triangles.

Table 6-43 — Specification of the triangulation type

triangle split code	Split
00	top-left to right bottom
01	bottom-left to top right
10	alternately top-left to bottom-right and bottom-left to top-right
11	alternately bottom-left to top-right and top-left to bottom-right

nr_of_mesh_nodes: This is a 16-bit unsigned integer defining the total number of nodes (vertices) of a (non-uniform) Delaunay mesh. These nodes include both interior nodes as well as boundary nodes.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

nr_of_boundary_nodes: This is a 10-bit unsigned integer defining the number of nodes (vertices) on the boundary of a (non-uniform) Delaunay mesh.

node0_x: This is a 13-bit signed integer specifying the x-coordinate of the first boundary node (vertex) of a mesh in half-pixel units with respect to a local coordinate system.

node0_y: This is a 13-bit signed integer specifying the y-coordinate of the first boundary node (vertex) of a mesh in half-pixel units with respect to a local coordinate system.

delta_x_len_vlc: This is a variable-length code specifying the length of the delta_x code that follows. The delta_x_len_vlc and delta_x codes together specify the difference between the x-coordinates of a node (vertex) and the previously encoded node (vertex). The definition of the delta_x_len_vlc and delta_x codes are given in Table B.33, the table for sprite motion trajectory coding.

delta_x: This is an integer that defines the value of the difference between the x-coordinates of a node (vertex) and the previously encoded node (vertex) in half pixel units. The number of bits in the bitstream for delta_x is delta_x_len_vlc.

delta_y_len_vlc: This is a variable-length code specifying the length of the delta_y code that follows. The delta_y_len_vlc and delta_y codes together specify the difference between the y-coordinates of a node (vertex) and the previously encoded node (vertex). The definition of the delta_y_len_vlc and delta_y codes are given in Table B.33, the table for sprite motion trajectory coding.

delta_y: This is an integer that defines the value of the difference between the y-coordinates of a node (vertex) and the previously encoded node (vertex) in half pixel units. The number of bits in the bitstream for delta_y is delta_y_len_vlc.

6.3.9.3 Mesh motion

motion_range_code: This is a 3-bit integer defined in Table 6-44. It specifies the dynamic range of motion vectors in half pel units.

Table 6-44 — motion range code

motion range code	motion vector range
1	[-32, 31]
2	[-64, 63]
3	[-128, 127]
4	[-256, 255]
5	[-512, 511]
6	[-1024, 1023]
7	[-2048, 2047]

node_motion_vector_flag: This is a 1 bit code specifying whether a node has a zero motion vector. When set to '1' it indicates that a node has a zero motion vector, in which case the motion vector is not encoded. When set to '0', it indicates the node has a nonzero motion vector and that motion vector data shall follow.

delta_mv_x_vlc: This is a variable-length code defining (together with delta_mv_x_res) the value of the difference in the x-component of the motion vector of a node compared to the x-component of a predicting motion vector. The definition of the delta_mv_x_vlc codes are given in Table B.12, the table for motion vector coding (MVD). The value delta_mv_x_vlc is given in half pixel units.

delta_mv_x_res: This is an integer which is used in mesh node motion vector decoding using an algorithm equivalent to that described in the section on video motion vector decoding, subclause 7.6.3. The number of bits in the bitstream for delta_mv_x_res is motion_range_code-1.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

delta_mv_y_vlc: This is a variable-length code defining (together with delta_mv_y_res) the value of the difference in the y-component of the motion vector of a node compared to the y-component of a predicting motion vector. The definition of the delta_mv_y_vlc codes are given in Table B.12, the table for motion vector coding (MVD). The value delta_mv_y_vlc is given in half pixel units.

delta_mv_y_res: This is an integer which is used in mesh node motion vector decoding using an algorithm equivalent to that described in the section on video motion vector decoding, subclause 7.6.3. The number of bits in the bitstream for delta_mv_y_res is motion_range_code-1.

6.3.10 FBA object

fba_object_start_code: The fba_object_start_code is the bit string '000001BA' in hexadecimal. It initiates a FBA object.

6.3.10.1 FBA object plane header

fba_object_plane_start_code: The fba_frame_start_code is the bit string '000001BB' in hexadecimal. It initiates a FBA object plane.

is_intra: This is a 1-bit flag which when set to '1' indicates that the FBA object is coded in intra mode. When set to '0' it indicates that the FBA object is coded in predictive mode.

fba_object_mask: This is a 2-bit integer defined in Table 6-45. It indicates whether FBA and BAP data are present in the FBA_frame.

Table 6-45 — FBA object mask

mask value	Meaning
00	unused
01	FAP present
10	BAP present
11	both FAP and BAP present

6.3.10.2 FBA object plane data

fap_quant: This is a 5-bit unsigned integer which is the quantiser scale used to compute the FAPi table step size or DCT fap_scale depending on the fba_object_coding_type. If the fba_object_coding_type is DCT this is a 5-bit unsigned integer used as the index to a fap_scale table for computing the quantisation step size of DCT coefficients. The value of fap_scale is specified in the following list:

fap_scale[0 - 31] = { 1, 1, 2, 3, 5, 7, 8, 10, 12, 15, 18, 21, 25, 30, 35, 42,
50, 60, 72, 87, 105, 128, 156, 191, 234, 288, 355, 439, 543, 674, 836, 1039}

fap_mask_type: This is a 2-bit integer. It indicates if the group mask will be present for the specified fap group, or if the complete faps will be present; its meaning is described in Table 6-46. In the case the type is '10' the '0' bit in the group mask indicates interpolate fap.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-46 — fap mask type

mask type	Meaning
00	no mask nor fap
01	group mask for copying
10	group mask for interpolation
11	fap

fap_group_mask[group_number]: This is a variable length bit entity that indicates, for a particular group_number which fap is represented in the bitstream. The value is interpreted as a mask of 1-bit fields. A 1-bit field in the mask that is set to '1' indicates that the corresponding fap is present in the bitstream. When that 1-bit field is set to '0' it indicates that the fap is not present in the bitstream. The number of bits used for the fap_group_mask depends on the group_number, and is given in Table 6-47.

Table 6-47 — fap group mask bits

group_number	No. of bits
1	2
2	16
3	12
4	8
5	4
6	5
7	3
8	10
9	4
10	4

NFAP[group_number] : This indicates the number of FAPs in each FAP group. Its values are specified in the following table:

Table 6-48 — NFAP definition

group_number	NFAP[group_number]
1	2
2	16
3	12
4	8
5	4
6	5
7	3
8	10
9	4
10	4

fba_suggested_gender: This is a 1-bit integer indicating the suggested gender for the face model. It does not bind the decoder to display a facial model of suggested gender, but indicates that the content would be more suitable for display with the facial model of indicated gender, if the decoder can provide one. If fba_suggested_gender is 1, the suggested gender is male, otherwise it is female.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fba_object_coding_type: This is a 1-bit integer indicating which coding method is used. Its meaning is described in Table 6-49.

Table 6-49 — fba_object_coding_type

type value	Meaning
0	predictive coding
1	DCT

is_i_new_max: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for I frame follows these 4, 1-bit fields.

is_i_new_min: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for I frame follows these 4, 1-bit fields.

is_p_new_max: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for P frame follows these 4, 1-bit fields.

is_p_new_min: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for P frame follows these 4, 1-bit fields.

bap_pred_quant_index: This is a 5-bit unsigned integer used as the index to a bap_pred_scale table for computing the quantisation step size of BAP values for predictive and DCT coding. If fba_object_coding_type is predictive, the value of bap_pred_scale is specified in the following list:

bap_pred_scale[0-31]=
{ 0, 1, 2, 3, 5, 7, 9, 11, 14, 17, 20, 23, 27, 31, 35, 39, 43, 47, 52, 57, 62, 67, 72, 77, 82, 88, 94, 100, 106, 113, 120, 127}

If the fba_object_coding_type is DCT this is a 5-bit unsigned integer used as the index to a bap_scale table for computing the quantisation step size of DCT coefficients. The value of bap_scale is specified in the following list:

bap_scale[0 – 31] = { 1, 1, 2, 3, 5, 7, 8, 10, 12, 15, 18, 21, 25, 30, 35, 42,
50, 60, 72, 87, 105, 128, 156, 191, 234, 288, 355, 439, 543, 674, 836, 1039}

bap_mask_type: This 2-bit value determines whether BAPs are transmitted individually or in groups.

bap_mask_type	Meaning
00	No mask nor BAPS
01	BAP group mask transmitted
10	reserved
11	All BAPs transmitted –no group mask

bap_group_mask: this is a variable-length mask indicating which BAPs in a group are present in the fba_object_plane.

group number	group name	No. of. bits
1	Pelvis	3
2	Left leg1	4
3	Right leg1	4
4	Left leg2	6

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5	Right leg2	6
6	Left arm1	5
7	Right arm1	5
8	Left arm2	7
9	Right arm2	7
10	Spine1	12
11	Spine2	15
12	Spine3	18
13	Spine4	18
14	Spine5	12
15	Left hand1	16
16	Right hand1	16
17	Left hand2	13
18	Right hand2	13
19	Global positioning	6
20	Extension1	22
21	Extension2	22
22	Extension3	22
23	Extension4	22
24	Extension5	22

bap_is_i_new_max: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for I frame follows these 4, 1-bit fields.

bap_is_i_new_min: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for I frame follows these 4, 1-bit fields.

bap_is_p_new_max: This is a 1-bit flag which when set to '1' indicates that a new set of maximum range values for P frame follows these 4, 1-bit fields.

bap_is_p_new_min: This is a 1-bit flag which when set to '1' indicates that a new set of minimum range values for P frame follows these 4, 1-bit fields.

6.3.10.3 Temporal Header

is_frame_rate: This is a 1-bit flag which when set to '1' indicates that frame rate information follows this bit field. When set to '0' no frame rate information follows this bit field.

is_time_code: This is a 1-bit flag which when set to '1' indicates that time code information follows this bit field. When set to '0' no time code information follows this bit field.

time_code: This is a 18-bit integer containing the following: time_code_hours, time_code_minutes, marker_bit and time_code_seconds as shown in Table 6-23. The parameters represent the hours, minutes, and seconds of the time base for the current object plane. The time code specifies the modulo part (i.e. the full second units) of the time base for the first object plane (in display order) after the GOV header. The time_code values in MPEG-4 are meant to be taken literally as precise multiples of exact seconds. It is not the same as SMPTE 12M time code.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-50 — Meaning of time_code

time_code	range of value	No. of bits	Mnemonic
time_code_hours	0 - 23	5	uimsbf
time_code_minutes	0 - 59	6	uimsbf
marker_bit	1	1	bslbf
time_code_seconds	0 - 59	6	uimsbf

skip_frames: This is a 1-bit flag which when set to '1' indicates that information follows this bit field that indicates the number of skipped frames. When set to '0' no such information follows this bit field.

6.3.10.4 Decode frame rate and frame skip

frame_rate: This is an 8 bit unsigned integer indicating the reference frame rate of the sequence.

seconds: This is a 4 bit unsigned integer indicating the fractional reference frame rate. The frame rate is computed as follows frame rate = (frame_rate + seconds/16).

frequency_offset: This is a 1-bit flag which when set to '1' indicates that the frame rate uses the NTSC frequency offset of 1000/1001. This bit would typically be set when frame_rate = 24, 30 or 60, in which case the resulting frame rate would be 23.97, 29.94 or 59.97 respectively. When set to '0' no frequency offset is present. I.e. if (frequency_offset ==1) frame rate = (1000/1001) * (frame_rate + seconds/16).

number_of_frames_to_skip: This is a 4-bit unsigned integer indicating the number of frames skipped. If the number_of_frames_to skip is equal to 15 (pattern "1111") then another 4-bit word follows allowing to skip up to 29 frames(pattern "11111110"). If the 8-bits pattern equals "11111111", then another 4-bits word will follow and so on, and the number of frames skipped is incremented by 30. Each 4-bit pattern of '1111' increments the total number of frames to skip with 15.

6.3.10.5 Decode new minmax

i_new_max[j]: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the I frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

i_new_min[j]: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the I frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

p_new_max[j]: This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the P frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

p_new_min[j]: This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the P frame. Arithmetic decoder max values must be non-negative and min values must be non-positive. Min values are positive in the bitstream and negated in decoder.

6.3.10.6 Decode viseme and expression

viseme_def: This is a 1-bit flag which when set to '1' indicates that the mouth FAPs sent with the viseme FAP may be stored in the decoder to help with FAP interpolation in the future.

init_face: This is a 1-bit flag which when set to '1' indicates that the neutral face may be modified within the neutral face constraints.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

expression_def: This is a 1-bit flag which when set to '1' indicates that the FAPs sent with the expression FAP may be stored in the decoder to help with FAP interpolation in the future.

6.3.10.7 Decode viseme_segment and expression_segment

viseme_segment_select1q[k]: This is the quantised value of viseme_select1 at frame k of a viseme FAP segment.

viseme_segment_select2q[k]: This is the quantised value of viseme_select2 at frame k of a viseme FAP segment.

viseme_segment_blendq[k]: This is the quantised value of viseme_blend at frame k of a viseme FAP segment.

viseme_segment_def[k]: This is a 1-bit flag which when set to '1' indicates that the mouth FAPs sent with the viseme FAP at frame k of a viseme FAP segment may be stored in the decoder to help with FAP interpolation in the future.

viseme_segment_select1q_diff[k]: This is the prediction error of viseme_select1 at frame k of a viseme FAP segment.

viseme_segment_select2q_diff[k]: This is the prediction error of viseme_select2 at frame k of a viseme FAP segment.

viseme_segment_blendq_diff[k]: This is the prediction error of viseme_blend at frame k of a viseme FAP segment.

expression_segment_select1q[k]: This is the quantised value of expression_select1 at frame k of an expression FAP segment.

expression_segment_select2q[k]: This is the quantised value of expression_select2 at frame k of an expression FAP segment.

expression_segment_intensity1q[k]: This is the quantised value of expression_intensity1 at frame k of an expression FAP segment

expression_segment_intensity2q[k]: This is the quantised value of expression_intensity2 at frame k of an expression FAP segment

expression_segment_select1q_diff[k]: This is the prediction error of expression_select1 at frame k of an expression FAP segment.

expression_segment_select2q_diff[k]: This is the prediction error of expression_select2 at frame k of an expression FAP segment.

expression_segment_intensity1q_diff[k]: This is the prediction error of expression_intensity1 at frame k of an expression FAP segment.

expression_segment_intensity2q_diff[k]: This is the prediction error of expression_intensity2 at frame k of an expression FAP segment.

expression_segment_init_face[k]: This is a 1-bit flag which indicates the value of init_face at frame k of an expression FAP segment.

expression_segment_def[k]: This is a 1-bit flag which when set to '1' indicates that the FAPs sent with the expression FAP at frame k of a viseme FAP segment may be stored in the decoder to help with FAP interpolation in the future.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.10.8 Decode i_dc, p_dc, and ac

dc_q: This is the quantised DC component of the DCT coefficients. For an intra FAP segment, this component is coded as a signed integer of either 16 bits or 31 bits. The DCT quantiser scales of the 68 FAPs are specified in the following list:

```
DCTQP[1 - 68] = {1,    1,    7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,
                 7.5,  7.5,  7.5,   15,    15,    15,    15,    5,     10,    10,
                 10,   10,   425,   425,   425,   425,   5,     5,     5,     5,
                 7.5,  7.5,  7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   20,    20,
                 20,   20,   10,    10,    10,    10,    255,   170,   255,   255,
                 7.5,  7.5,  7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,   7.5,
                 15,   15,   15,    15,    10,    10,    10,    10}
```

For DC coefficients, the quantisation step size is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i] ÷ 3.0

dc_q_diff: This is the quantised prediction error of a DC coefficient of an inter FAP segment. Its value is computed by subtracting the decoded DC coefficient of the previous FAP segment from the DC coefficient of the current FAP segment. It is coded by a variable length code if its value is within [-255, +255]. Outside this range, its value is coded by a signed integer of 16 or 32 bits.

count_of_runs: This is the run length of zeros preceding a non-zero AC coefficient.

ac_q[i][next]: This is a quantised AC coefficients of a segment of FAPi. For AC coefficients, the quantiser step size is three times larger than the DC quantisation step size and is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i]

BAPs are decoded using the same process. For BAPs, DCT quantiserscales (DCTQP[i]) have the same value as BAP predictive coding step sizes as defined in Annex C.

6.3.10.9 Decode bap min max

bap_i_new_max[i] – This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the I frame.

bap_i_new_min[i] – This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the I frame.

bap_p_new_max[i] – This is a 5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the P frame.

bap_p_new_min[i] – This is a 5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the P frame.

6.3.11 3D Mesh Object

6.3.11.1 3D_Mesh_Object

3D_MO_start_code: This is a unique 16-bit code that is used for synchronization purpose. The value of this code is always ‘0000 0000 0010 0000’.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.11.2 3D_Mesh_Object_Layer

3D_MOL_start_code: This is a unique 16-bit code that is used for synchronization purposes. The value of this code is always '0000 0000 0011 0000'.

mol_id: This 8-bit unsigned integer specifies a unique id for the mesh object layer. Value 0 indicates a base layer, and value larger than 0 a refinement layer. The first 3D_Mesh_Object_Layer immediately after a 3D_Mesh_Object_Header must have mold_id=0, and subsequent 3DMesh_Object_Layer's within the same 3D_Mesh_Object must have mold_id>0.

ce_SNHC_n_vertices: This is the number of vertices in the current resolution of the 3D mesh. Used to support computational graceful degradation.

ce_SNHC_n_triangles: This is the number of triangles in the current resolution of the 3D mesh. Used to support computational graceful degradation.

ce_SNHC_n_edges: This is the number of edges in the current resolution of the 3D mesh. Used to support computational graceful degradation.

6.3.11.3 3DMesh_Object_Base_Layer

3D_MOBL_start_code: This is a code of length 16 that is used for synchronization purposes. It also indicates three different partition types for error resilience.

Table 6-51 — Definition of partition type information

3D_MOBL_start_code	partition type	Meaning
'0000 0000 0011 0001'	partition_type_0	One or more groups of vg, tt and td.
'0000 0000 0011 0011'	partition_type_1	One or more vgs
'0000 0000 0011 0100'	partition_type_2	One pair of tt and td.

mobl_id: This 8-bit unsigned integer specifies a unique id for the mesh object component.

one_bit: This boolean value is always true. This value is used for byte alignment.

last_component: This boolean value indicates if there are more connected components to be decoded. If **last_component** is '1', then the last component has been decoded. Otherwise there are more components to be decoded. This field is arithmetic coded

codap_last_vg: This boolean value indicates if the current vg is the last one in the partition. The value is false if there are more vgs to be decoded in the partition.

codap_vg_id: This unsigned integer indicates the id of the vertex graph corresponding to the current simple polygon in partition_type_2. The length of this value is a log scaled value of the vg_number of vg decoded from the previous partition_type_1. If there is only one vg in the previous partition_type_1.

codap_left_bloop_idx: This unsigned integer indicates the left starting index, within the bounding loop table of a connected component, for the triangles that are to be reconstructed in a partition. The length of this value is the log scaled value of the size of the bounding loop table.

codap_right_bloop_idx: This unsigned integer indicates the right starting index, within the bounding loop table of a connected component, for the triangles that are to be reconstructed in a partition. The length of this value is the log scaled value of the size of the bounding loop table.

codap_bdry_pred: This boolean value denotes how to predict geometry and photometry information that are in common with two or more partitions. If **codap_bdry_pred** is '1', the restricted boundary prediction mode is used, otherwise, the extended boundary prediction mode is used.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.11.4 3DMesh_Object_Header

ccw: This boolean value indicates if the vertex ordering of the decoded faces follows a counter clock-wise order.

convex: This boolean value indicates if the model is convex.

solid: This boolean value indicates if the model is solid.

creaseAngle: This 6-bit unsigned integer indicates the crease angle.

6.3.11.5 coord_header

coord_binding: This 2 bit unsigned integer indicates the binding of vertex coordinates to the 3D mesh. Table 6-52 shows the admissible values for **coord_binding**.

Table 6-52 — Admissible values for coord_binding

coord_binding	binding
00	forbidden
01	bound_per_vertex
10	forbidden
11	forbidden

coord_bbox: This boolean value indicates whether a bounding box is provided for the geometry. If no bounding box is provided, a default bounding box is used. The default bounding box is defined as **coord_xmin**=0, **coord_ymin**=0, **coord_zmin**=0, and **coord_size**=1.

coord_xmin, **coord_ymin**, **coord_zmin**: These floating point values indicates the lower left corner of the bounding box in which the geometry lies.

coord_size: This floating point value indicates the size of the bounding box.

coord_quant: This 5-bit unsigned integer indicates the quantisation step used for geometry. The minimum value of coord_quant is 1 and the maximum is 24.

coord_pred_type: This 2-bit unsigned integer indicates the type of prediction used to reconstruct the vertex coordinates of the mesh. Table 6-53 shows the admissible values for coord_pred_type.

Table 6-53 — Admissible values for coord_pred_type

coord_pred_type	prediction type
00	no_prediction
01	forbidden
10	parallelogram_prediction
11	reserved

coord_nlambda: This 2-bit unsigned integer indicates the number of ancestors used to predict geometry. The only admissible value of **coord_nlambda** is 3. Table 6-54 shows the admissible values as a function of **coord_pred_type.**

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-54 — Admissible values for coord_nlambda as a function of coord_prediction type

coord_pred_type	coord_nlambda
00	not coded
10	3

coord_lambda: This signed fixed-point number indicates the weight given to an ancestor for prediction. The number of bits used for this field is equal to **coord_quant** + 3. The 3 leading bits represent the integer part, and the **coord_quant** remaining bits the fractional part.

6.3.11.6 normal_header

normal_binding: This 2 bit unsigned integer indicates the binding of normals to the 3D mesh. The admissible values are described in Table 6-55.

Table 6-55 — Admissible values for normal_binding

normal_binding	binding
00	not_bound
01	bound_per_vertex
10	bound_per_face
11	bound_per_corner

normal_bbox: This boolean value should always be false (‘0’).

normal_quant: This 5-bit unsigned integer indicates the quantisation step used for normals. The minimum value of normal_quant is 3 and the maximum is 31.

normal_pred_type: This 2-bit unsigned integer indicates how normal values are predicted. Table 6-56 shows the admissible values, and Table 6-57 shows admissible values as a function of normal_binding.

Table 6-56 — Admissible values for normal_pred_type

normal_pred_type	prediction type
00	no_prediction
01	tree_prediction
10	parallelogram_prediction
11	reserved

Table 6-57 — Admissible combinations of normal_binding and normal_pred_type

normal_binding	normal_pred_type
not_bound	not coded
bound_per_vertex	no_prediction, parallelogram_prediction
bound_per_face	no_prediction, tree_prediction
bound_per_corner	no_prediction, tree_prediction

normal_nlambda: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **normal_nlambda** are 1, 2, and 3. Table 6-58 shows admissible values as a function of **normal_pred_type.**

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-58 — Admissible values for normal_nlambda as a function of normal_prediction type

normal_pred_type	normal_nlambda
no_prediction	not coded
tree_prediction	1, 2, 3
parallelogram_prediction	3

normal_lambda: This signed fixed-point indicates the weight given to an ancestor for prediction. The number of bits used for normal_lambda is (normal_quant-3)/2+3. The 3 leading bits represent the integer part, and the **normal_quant** remaining bits the fractional part.

6.3.11.7 color_header

color_binding: This 2 bit unsigned integer indicates the binding of colors to the 3D mesh. Table 6-59 shows the admissible values.

Table 6-59 — Admissible values for color_binding

color_binding	Binding
00	not_bound
01	bound_per_vertex
10	bound_per_face
11	bound_per_corner

color_bbox: This boolean indicates if a bouding box for colors is given. If no bounding box is provided, a default bounding box is used. The default bounding box is defined as **color_rmin**=0, **color_gmin**=0, **color_bmin**=0, and **color_size**=1.

color_rmin, **color_gmin**, **color_bmin**: These floating point values give the position of the lower left corner of the bounding box in RGB space.

color_size: This floating point value gives the size of the color bounding box.

color_quant: This 5-bit unsigned integer indicates the quantisation step used for colors. The minimum value of color_quant is 1 and the maximum is 16.

color_pred_type: This 2-bit unsigned integer indicates how colors are predicted. Table 6-60 shows the admissible values, and Table 6-61 shows admissible values as a function of color_binding.

Table 6-60 — Admissible values for color_pred_type

color_pred_type	prediction type
00	no_prediction
01	tree_prediction
10	parallelogram_prediction
11	reserved

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-61 — Admissible combinations of color_binding and color_pred_type

color_binding	color_pred_type
not_bound	not coded
bound_per_vertex	no_prediction, parallelogram_prediction
bound_per_face	no_prediction, tree_prediction
bound_per_corner	no_prediction, tree_prediction

color_nlambda: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **color_nlambda** are 1, 2, and 3. Table 6-62 shows admissible values as a function of **normal_pred_type.**

Table 6-62 — Admissible values for color_nlambda as a function of color_prediction type

color_pred_type	color_nlambda
no_prediction	not coded
tree_prediction	1, 2, 3
parallelogram_prediction	3

color_lambda: This signed fixed-point indicates the weight given to an ancestor for prediction. The number of bits used for this field is equal to **color_quant** + 3. The 3 leading bits represent the integer part, and the **normal_quant** remaining bits the fractional part.

6.3.11.8 texCoord_header

texCoord_binding: This 2 bit unsigned integer indicates the binding of texture coordinates to the 3D mesh. Table 6-63 describes the admissible values.

Table 6-63 — Admissible values for texCoord_binding

texCoord_binding	Binding
00	not_bound
01	bound_per_vertex
10	forbidden
11	bound_per_corner

texCoord_bbox: This boolean value indicates if a bounding box for texture coordinates is given. If no bounding box is provided, a default bounding box is used. The default bounding box is defined as **texCoord_umin**=0, **texCoord_vmin**=0, and **texCoord_size**=1.

texCoord_umin, **texCoord_vmin**: These floating point values give the position of the lower left corner of the bounding box in 2D space.

texCoord_size: This floating point value gives the size of the texture coordinate bounding box.

texCoord_quant: This 5-bit unsigned integer indicates the quantisation step used for texture coordinates. The minimum value of texCoord_quant is 1 and the maximum is 16.

texCoord_pred_type: This 2-bit unsigned integer indicates how colors are predicted. Table 6-64 shows the admissible values, and Table 6-65 shows admissible values as a function of texCoord_binding.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-64 — Admissible values for texCoord_pred_type

texCoord_pred_type	prediction type
00	no_prediction
01	forbidden
10	parallelogram_prediction
11	reserved

Table 6-65 — Admissible combinations of texCoord_binding and texCoord_pred_type

texCoord_binding	texCoord_pred_type
not_bound	not coded
bound_per_vertex	no_prediction, parallelogram_prediction
bound_per_corner	no_prediction, tree_prediction

texCoord_nlambda: This 2-bit unsigned integer indicates the number of ancestors used to predict normals. Admissible values of **texCoord_nlambda** are 1, 2, and 3. Table 6-66 shows admissible values as a function of **texCoord_pred_type.**

Table 6-66 — Admissible values for texCoord_nlambda as a function of texCoord_prediction type

texCoord_pred_type	texCoord_nlambda
not_prediction	not coded
tree_prediction	1, 2, 3
parallelogram_prediction	3

texCoord_lambda: This signed fixed-point indicates the weight given to an ancestor for prediction. The number of bits used for this field is equal to **texCoord_quant** + 3. The 3 leading bits represent the integer part, and the **texCoord_quant** remaining bits the fractional part.

6.3.11.9 ce_SNHC_header

ce_SNHC_n_proj_surface_spheres: The number of Projected Surface Spheres. Typically, this number is equal to 1.

ce_SNHC_x_coord_center_point: The x-coordinate (in 32-bit IEEE floating point format) of the center point (typically the gravity point of the object) of the Projected Surface Sphere.

ce_SNHC_y_coord_center_point: The y-coordinate (in 32-bit IEEE floating point format) of the center point (typically the gravity point of the object) of the Projected Surface Sphere.

ce_SNHC_z_coord_center_point: The z-coordinate (in 32-bit IEEE floating point format) of the center point (typically the gravity point of the object) of the Projected Surface Sphere.

ce_SNHC_normalized_screen_distance_factor: This indicates where the virtual screen is placed, compared to the radius of the Projected Surface Sphere. The distance between the center point of the Projected Surface Sphere and the virtual screen is equal to *ce_SNHC_radius*/*(ce_SNHC_normalized_screen_distance_factor+1)*. Note that *ce_SNHC_radius* is specified for each Projected Surface Sphere, while *ce_SNHC_normalized_screen_distance_factor* is specified only once.

ce_SNHC_radius: The radius (in 32-bit IEEE floating point format) of the Projected Surface Sphere.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ce_SNHC_min_proj_surface: The minimal projected surface value (in 32-bit IEEE floating point format) on the corresponding Projected Surf ace Sphere. This value is often (but not necessarily) equal to one of the *ce_SNHC_proj_surface* values.

ce_SNHC_n_proj_points: The number of points on the Projected Surface Sphere in which the projected surface will be transmitted. For all other points, the projected surface is determined by linear interpolation. *ce_SNHC_n_proj_points* is typically small (e.g. 20) for the first Projected Surface Sphere and very small (e.g. 3) for additional Projected Surface Spheres.

ce_SNHC_sphere_point_coord: This indicates the index of the point position in a octahedron, as explained in "inverse quantisation" section (see subclause 7.13.8.4).

ce_SNHC_proj_surface: The projected surface (in 32-bit IEEE floating point format) in the point specified by *ce_SNHC_sphere_point_coord*.

6.3.11.10 connected component

has_stitches: This boolean value indicates if stitches are applied for the current connected component (within itself or between the current component and connected components previously decoded). This field is arithmetic coded.

6.3.11.11 vertex_graph

vg_simple: This boolean value indicates if the current vertex graph is simple. A simple vertex graph does not contain any loop. This field is arithmetic coded.

vg_last: This boolean value indicates if the current run is the last run starting from the current branching vertex. This field is not coded for the first run of each branching vertex, i.e. when the skip_last variable is true. When not coded the value of **vg_last** for the current vertex run is considered to be false. This field is arithmetic coded.

vg_forward_run: This boolean value indicates if the current run is a new run. If it is not a new run, it is a previously traversed run, indicating a loop in the graph. This field is arithmetic coded.

vg_loop_index: This unsigned integer indicates the index of run to which the current loop connects. Its unary representation (see Table 6-67) is arithmetic coded. If the variable openloops is equal to **vg_loop_index**, the trailing '1' in the unary representation is omitted.

Table 6-67 — Unary representation of the vg_loop_index field

vg_loop_index	unary representation
0	1
1	01
2	001
3	0001
4	00001
5	000001
6	0000001
...	
openloops-1	openloops-1 0's

vg_run_length: This unsigned integer indicates the length of the current vertex run. Its unary representation (see Table 6-68) is arithmetic coded.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-68 — Unary representation of the vg_run_length field

vg_run_length	unary representation
1	1
2	01
3	001
4	0001
5	00001
6	000001
7	0000001
8	00000001
N	n-1 0's followed by 1

vg_leaf: This boolean value indicates if the last vertex of the current run is a leaf vertex. If it is not a leaf vertex, it is a branching vertex. This field is arithmetic coded.

vg_loop: This boolean value indicates if the leaf of the current run connects to a branching vertex of the graph, indicating a loop. This field is arithmetic coded.

6.3.11.12 stitches

stitch_cmd: This boolean value indicates if a stitching command of the type PUSH, POP or GET is associated to the current vertex. This field is arithmetic coded.

stitch_pop_or_get: This boolean value indicates if a stitching command of the type POP or GET is associated to the current vertex. This field is arithmetic coded.

stitch_pop: This boolean value indicates if a stitching command of the type POP is associated to the current vertex. This field is arithmetic coded.

stitch_stack_index: This unsigned integer value indicates the depth in the anchor stack where the anchor which the current vertex will be stitched to is located. This field is arithmetic coded.

stitch_incr_length: This integer value indicates the incremental length of the current stitch that must be added or subtracted to the length that is currently stored at the anchor. This field is arithmetic coded.

stitch_incr_length_sign: This boolean value indicates if the stitch_incr_length is negative. This field is arithmetic coded.

stitch_push: This boolean value indicates if the current vertex must be pushed into the stack of anchors. This field is arithmetic coded.

stitch_reverse: This boolean value indicates whether the current vertex must be stitched to its anchor using a reverse stitch as opposed to a forward stitch which is the default behavior. This field is arithmetic coded.

stitch_length: This unsigned integer value. This field is arithmetic coded.

6.3.11.13 triangle_tree

branch_position: This integer variable is used to store the last branching triangle in a partition.

tt_run_length: This unsigned integer indicates the length of the current triangle run. Its unary representation (see Table 6-69) is arithmetic coded.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-69 — Unary representation of the tt_run_length field

tt_run_length	unary representation
1	1
2	01
3	001
4	0001
5	00001
6	000001
7	0000001
8	00000001
N	n-1 0's followed by 1

tt_leaf: This boolean value indicates if the last triangle of the current run is a leaf triangle. If it is not a leaf triangle, it is a branching triangle. This field is arithmetic coded.

triangulated: This boolean value indicates if the current component contains triangles only. This field is arithmetic coded.

marching_triangle: This boolean value is determined by the position of the triangle in the triangle tree. If **marching_triangle** is 0, the triangle is a leaf or a branch. Otherwise, the triangle is a run.

marching_edge: This boolean value indicates the marching edge of an edge inside a triangle run. If **marching_edge** is false, it stands for a march to the left, otherwise it stands for a march to the right. This field is arithmetic coded.

polygon_edge: This boolean value indicates whether the base of the current triangle is an edge that should be kept when reconstructing the 3D mesh object. If the base of the current triangle is not kept, the edge is discarded. This field is arithmetic coded.

codap_branch_len: This unsigned integer indicates the length of the next branch to be traversed. The length of this value is the log scaled value of the size of the bounding loop table.

6.3.11.14 triangle

td_orientation: This boolean value informs the decoder the traversal order of tt/td pair at a branch. This field is arithmetic coded. Table 6-70 shows the admissible values.

Table 6-70 — Admissible values for td_orientation

td_orientation	traversal order
0	right branch first
1	left branch first

visited: This variable indicates if the current vertex has been visited or not. When **codap_bdry_pred** is '1', visited is true for the vertices visited in the current partition. However, when **codap_bdry_pred** is '0', visited is true for the vertices visited in the previous partitions as well as in the current partition.

vertex_index: This variable indicates the index of the current vertex in the vertex array.

no_ancestors: This boolean value is true if there are no ancestors to use for prediction of the current vertex.

coord_bit: This boolean value indicates the value of a geometry bit. This field is arithmetic coded.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

coord_leading_bit: This boolean value indicates the value of a leading geometry bit. This field is arithmetic coded.

coord_sign_bit: This boolean value indicates the sign of a geometry sample. This field is arithmetic coded.

coord_trailing_bit: This boolean value indicates the value of a trailing geometry bit. This field is arithmetic coded.

normal_bit: This boolean value indicates the value of a normal bit. This field is arithmetic coded.

normal_leading_bit: This boolean value indicates the value of a leading normal bit. This field is arithmetic coded.

normal_sign_bit: This boolean value indicates the sign of a normal sample. This field is arithmetic coded.

normal_trailing_bit: This boolean value indicates the value of a trailing normal bit. This field is arithmetic coded.

color_bit: This boolean value indicates the value of a color bit. This field is arithmetic coded.

color_leading_bit: This boolean value indicates the value of a leading color bit. This field is arithmetic coded.

color_sign_bit: This boolean value indicates the sign of a color sample. This field is arithmetic coded.

color_trailing_bit: This boolean value indicates the value of a trailing color bit. This field is arithmetic coded.

texCoord_bit: This boolean value indicates the value of a texture bit. This field is arithmetic coded.

texCoord_leading_bit: This boolean value indicates the value of a leading texture bit. This field is arithmetic coded.

texCoord_sign_bit: This boolean value indicates the sign of a texture sample. This field is arithmetic coded.

texCoord_trailing_bit: This boolean value indicates the value of a trailing texture bit. This field is arithmetic coded.

6.3.11.15 3DMeshObject_Refinement_Layer

3D_MORL_start_code: This is a unique 16-bit code that is used for synchronization purpose. The value of this code is always '0000 0000 0011 0010'.

morl_id: This 8-bit unsigned integer specifies a unique id for the forest split component.

connectivity_update: This 2-bit variable indicates whether the forest split operation results in a refinement of the connectivity of the mesh or not.

Table 6-71 — Admissible values for connectivity_update

connectivity_update	meaning
00	not_updated
01	fs_update
10	reserved
11	reserved

pre_smoothing: This boolean value indicates whether the current forest split operation uses a pre-smoothing step to globally predict vertex positions.

post_smoothing: This boolean value indicates whether the current forest split operation uses a post-smoothing step to remove quantisation artifacts.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

stuffing_bit: This boolean value is always true.

other_update: This boolean value indicates whether updates for vertex coordinates and properties associated with faces and corners not incident to any tree of the forest follow in the bitstream or not.

other_update: this boolean value indicates whether updates for vertex coordinates and properties associated with faces and corners not incident to any tree of the forest follow in the bitstream or not.

6.3.11.15.1 pre_smoothing_parameters

pre_smoothing_n: This integer value indicates the number of iterations of the pre-smoothing filter.

pre_smoothing_lambda: This float value is the first parameter of the pre-smoothing filter.

pre_smoothing_mu: This float value is the second parameter of the pre-smoothing filter.

6.3.11.15.2 post_smoothing_parameters

post_smoothing_n: This integer value indicates the number of iterations of the pre-smoothing filter.

post_smoothing_lambda: This float value is the first parameter of the pre-smoothing filter.

post_smoothing_mu: This float value is the second parameter of the pre-smoothing filter.

6.3.11.15.3 fs_pre_update

pfs_forest_edge: This boolean value indicates if an edge should be added to the forest built so far.

6.3.11.15.4 smoothing_constraints

smooth_with_sharp_edges: This boolean value indicates if data is included in the bitstream to mark smoothing discontinuity edges or not. If **sharp_edges**==0 no edge is marked as a smoothing discontinuity edge. If smoothing discontinuity edges are marked, then both the pre-smoothing and post-smoothing filters take them into account.

smooth_with_fixed_vertices: This boolean value indicates if data is included in the bitstream to mark vertices which do not move during the smoothing process. If **smooth_with_fixed_vertices**==0 all vertices are allowed to move. If fixed vertices are marked, then both the pre-smoothing and post-smoothing filters take them into account.

smooth_sharp_edge: This boolean value indicates if a corresponding edge is marked as a smoothing discontinuity edge.

smooth_fixed_vertex: This boolean value indicates if a corresponding vertex is marked as a fixed vertex or not.

6.3.12 Upstream message

6.3.12.1 upstream_message

upstream_message_type: This 3-bit value indicates the type of the upstream information as shown in Table 6-72. The meaning of each upstream type is as follows:

video_newpred: This upstream message conveys the decoding status of the receiver (decoder) for the NEWPRED mode. The NEWPRED is the error resilience tool by selecting the reference picture of the inter-frame coding according to the error condition of the network. This upstream message shows whether the decoder decode the forward video data correctly or not. This message returns corresponding to the VOP or Video Packet of the forward video data. Which type of message is required for the encoder is indicated in requested_upstream_message_type in the VOL header of the downstream data. The decoder definitions of NEWPRED are described in subclauses 7.14 and E.1.6.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SNHC_QoS: This upstream message conveys some information that reveals to the encoder (server) the performances of the decoder w.r.t. decoding and rendering 3D objects with different parameter settings, i.e. with a varying number of triangles, different screen coverages of the rendered objects and different rendering modes. Using this information, the encoder can restrict the 3D content to the capabilities of the decoder, using for instance mesh simplification and/or rendering mode selection.

Table 6-72 — Meaning of upstream_message_type

upstream_message_type	meaning
000	reserved
001	video_newpred
010	SNHC_QoS
011-111	reserved

6.3.12.2 upstream_video_newpred

newpred_upstream_message_type: This indicates whether the corresponding NP segment is correctly decoded or not. Which type of message is required for the encoder is indicated in requested_upstream_message_type in the VOL header of the downstreamdata. In the other case, this indicates requesting intra refresh.

00: NP_NACK. It indicates the erroneous decoding of the NP segment.
01: NP_ACK. It indicates the correct decoding of the NP segment.
10: Intra refresh command.
11: Reserved.

unreliable_flag: This field presents only if newpred_upstream_message_type is 'NP_NACK'. The unreliable_flag is set to 1 when a reliable value of vop_id is not available at the decoder. (When the NP segment is erred, a reliable vop_id may not be available at the decoder. On the other hand, a reliable vop_id is available, when the decoder cannot decode due to the luck of the reference picture.)

0: reliable
1: unreliable

vop_id: When the newpred_upstream_message_type is 'NP_NACK' or 'NP_ACK', this indicates the ID of VOP which is incremented by 1 whenever a VOP is encoded. The vop_id is copied from the vop_id field of the NP segment header in the corresponding forward channel data when the reliable vop_id is available. Otherwise, it may happen in the case of NP_NACK that the vop_id is incremented by 1 from the reliable vop_id of the previously received NP segment in the same location of the current NP segment.

When the newpred_upstream_message_type is 'Intra refresh command', this indicates the ID of Intra refresh which is incremented by 1 whenever new refresh is required. The length of this field is 4 bits in the Intra refresh case. In the case that Intra refresh command is repeatedly returned for the same error until the proper action corresponding to the previous Intra refresh command reaches, this ID is set to the same number as the previous Intra refresh command.

macroblock_number: The macroblock_number is the macroblock address of the start of the corresponding NP segment or the refresh area.

end_macroblock_number: This field is present only when the newpred_upstream_message_type is 'Intra refresh command'. The end_macroblock_number is the macroblock address of the end of the refresh area.

requested_vop_id_for_prediction: This field is present only if newpred_upstream_message_type is 'NP_NACK'. This indicates the requested vop_id of the NP segment for reference by the decoder. Typically it is the vop_id of the last correctly decoded NP segment in the same location of the current NP segment.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.12.3 upstream_SNHC_QoS

screen_width: Screen width used during the calibration process. screen_width is expressed in number of pixels.

screen_height: Screen height used during the calibration process. screen_height is expressed in number of pixels.

n_rendering_modes: n_rendering_modes is the number of rendering modes for which information is transmitted.

rendering_mode_type: rendering_mode_type is the kind of rendering used during the calibration process for a particular performance curve. The different rendering types are coded according to the following table.

Table 6-73 — Meaning of rendering_mode_type

rendering_mode_type	Rendering mode
0000	Wire-framed
0001	Flat shading
0010	Smooth shading
0011	Texture rendering
0100-1111	Reserved for later use

n_curves: n_curves is the number of performance curves transmitted for one particular rendering mode.

triangle_parameter: triangle_parameter is the number of triangles in units of 64 triangles.

n_points_on_curve: n_points_on_curve is the number of points specified for one particular performance curve.

screen_coverage_parameter: screen_coverage_parameter is the number of pixels, expressed in percentage of the screen size. 0x00 corresponds to 0%, while 0xFF corresponds to 100%. All other points are determined by linear interpolation.

frame_rate_value: frame_rate_value is the frame rate for one particular point on one particular curve. For achieving enough precision, 12 bits are used. The 8 Most Significant Bits represent the integer value, the 4 Least Significant Bits represent the fractional value.

6.3.13 Studio Video Object

6.3.13.1 Studio Visual Object

visual_object_start_code: The visual_object_start_code is the bit string '000001B5' in hexadecimal. It identifies the beginning of a visual object header.

visual_object_verid: This is a 4-bit code which identifies the version number of the visual object. Its meaning is defined in Table 6-74.

Table 6-74 — Meaning of visual_object_verid

visual_object_verid	Meaning
0000	reserved
0001	object type listed in Table 9.1
0010	object type listed in Table 9.2
0011	reserved
0100	object type listed in Table 9.3
0101 – 1111	reserved

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

visual_object_type: The visual_object_type is a 4-bit code given in Table 6-75 which identifies the type of the visual object.

Table 6-75 — Meaning of visual object type

code	visual object type
0000	reserved
0001	video ID
0010	still texture ID
0011	mesh ID
0100	face ID
0101	reserved
:	:
:	:
1111	reserved

video_object_start_code: The video_object_start_code is a string of 32 bits. The first 27 bits are '0000 0000 0000 0000 0000 0001 000' in binary and the last 5-bits represent one of the values in the range of '00000' to '11111' in binary. The video_object_start_code identifies the beginning of a video object header.

6.3.13.2 Extension and user data Studio

6.3.13.2.1 Extension data

extension_start_code: The extension_start_code is the bit string '000001B8' in hexadecimal. It identifies the beginning of extensions.

6.3.13.2.2 User data Studio

user_data_start_code: The user_data_start_code is the bit string '000001B2' in hexadecimal. It identifies the beginning of user data. The user data continues until receipt of another start code.

user_data: This is an 8 bit integer, an arbitrary number of which may follow one another. User data is defined by users for their specific applications. In the series of consecutive user_data bytes there shall not be a string of 23 or more consecutive zero bits.

6.3.13.2.3 Sequence display extension

This specification does not define the display process. The information in this extension does not affect the decoding process and may be ignored by decoders that conform to this specification.

extension_start_code_identifier: This is a 4-bit integer which identifies the extension. See Table 6-4.

video_format: This is a three bit integer indicating the representation of the pictures before being coded in accordance with this specification. Its meaning is defined in Table 6-76. If the sequence_display_extension() is not present in the bitstream then the video format may be assumed to be "Unspecified video format".

--`,,`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-76 — Meaning of video_format

video_format	Meaning
000	component
001	PAL
010	NTSC
011	SECAM
100	MAC
101	Unspecified video format
110	reserved
111	reserved

video_range: This one-bit flag indicates the black level and range of the luminance and chrominance signals. In the case that sequence_display_extension() is not present in the bitstream, video_range is assumed to have the value 0 (a range of Y from 16 to 235 for 8-bit video).

colour_description: A flag which if set to '1' indicates the presence of colour_primaries, transfer_characteristics and matrix_coefficients in the bitstream.

colour_primaries: This 8-bit integer describes the chromaticity coordinates of the source primaries, and is defined in Table 6-77.

Table 6-77 — Colour Primaries

Value	Primaries
0	(forbidden)
1	Recommendation ITU-R BT.709 primary x y green 0,300 0,600 blue 0,150 0,060 red 0,640 0,330 white D65 0,3127 0,3290
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	Recommendation ITU-R BT.470-2 System M primary x y green 0,210,71 blue 0,140,08 red 0,670,33 white C 0,310 0,316
5	Recommendation ITU-R BT.470-2 System B, G primary x y green 0,290,60 blue 0,150,06 red 0,640,33 white D65 0,313 0,329

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6	SMPTE 170M primary x y green 0,310 0,595 blue 0,155 0,070 red 0,630 0,340 white D65 0,3127 0,3290
7	SMPTE 240M (1987) primary x y green 0,310 0,595 blue 0,155 0,070 red 0,630 0,340 white D65 0,3127 0,3291
8	Generic film (colour filters using Illuminant C) primary x y green 0,243 0,692 (Wratten 58) blue 0,145 0,049 (Wratten 47) red 0,681 0,319 (Wratten 25)
9-255	reserved

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero the chromaticity is assumed to be that corresponding to colour_primaries having the value 1.

transfer_characteristics: This 8-bit integer describes the opto-electronic transfer characteristic of the source picture, and is defined in Table 6-78.

Table 6-78 — Transfer Characteristics

Value	Transfer Characteristic
0	(forbidden)
1	Recommendation ITU-R BT.709 $V = 1{,}099\ L_C^{0{,}45} - 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	Recommendation ITU-R BT.470-2 System M Assumed display gamma 2,2
5	Recommendation ITU-R BT.470-2 System B, G Assumed display gamma 2,8
6	SMPTE 170M $V = 1{,}099\ L_C^{0{,}45} - 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

7	SMPTE 240M (1987) $V = 1{,}1115\ L_C^{0{,}45} - 0{,}1115$ for $L_C \geq 0{,}0228$ $V = 4{,}0\ L_C$ for $0{,}0228 > L_C$
8	Linear transfer characteristics i.e. $V = L_C$
9	Logarithmic transfer characteristic (100:1 range) $V = 1.0\text{-}\mathrm{Log}_{10}(Lc)/2$ for $1 = L_C = 0.01$ $V = 0.0$ for $0.01 > L_C$
10	Logarithmic transfer characteristic (316.22777:1 range) $V = 1.0\text{-}\mathrm{Log}_{10}(Lc)/2.5$ for $1 = L_C = 0.0031622777$ $V = 0.0$ for $0.0031622777 > L_C$
11-255	reserved

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero the transfer characteristics are assumed to be those corresponding to transfer_characteristics having the value 1.

matrix_coefficients: This 8-bit integer describes the matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries, and is defined in Table 6-79.

In this table:

E'_Y is analogue with values between 0 and 1

E'_{PB} and E'_{PR} are analogue between the values -0,5 and 0,5

E'_R, E'_G and E'_B are analogue with values between 0 and 1

White is defined as $E'_y = 1$, $E'_{PB} = 0$, $E'_{PR} = 0$; $E'_R = E'_G = E'_B = 1$.

Y, Cb and Cr are related to E'_Y, E'_{PB} and E'_{PR} by the following formulae:

if **video_range=0:**

$$Y = (219 * 2^{n-8} * E'_Y) + 2^{n-4}$$
$$Cb = (224 * 2^{n-8} * E'_{PB}) + 2^{n-1}$$
$$Cr = (224 * 2^{n-8} * E'_{PR}) + 2^{n-1}$$

if **video_range=1:**

$$Y = ((2^n -1) * E'_Y)$$
$$Cb = ((2^n -1) * E'_{PB}) + 2^{n-1}$$
$$Cr = ((2^n -1) * E'_{PR}) + 2^{n-1}$$

for n bit video.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For example, for 8 bit video,

video_range=0 gives a range of Y from 16 to 235, Cb and Cr from 16 to 240;

video_range=1 gives a range of Y from 0 to 255, Cb and Cr from 0 to 255.

NOTE The decoding process given by this specification limits output sample values for Y, Cr and Cb to the range [0:255]. Thus sample values outside the range implied by the above equations may occasionally occur at the output of the decoding process. In particular the sample values 0 and 255 may occur.

Table 6-79 — Matrix Coefficients

Value	Matrix
0	(forbidden)
1	Recommendation ITU-R BT.709 $E'_Y = 0{,}7154\ E'_G + 0{,}0721\ E'_B + 0{,}2125\ E'_R$ $E'_{PB} = -0{,}386\ E'_G + 0{,}500\ E'_B - 0{,}115\ E'_R$ $E'_{PR} = -0{,}454\ E'_G - 0{,}046\ E'_B + 0{,}500\ E'_R$
2	Unspecified Video Image characteristics are unknown.
3	reserved
4	FCC $E'_Y = 0{,}59\ E'_G + 0{,}11\ E'_B + 0{,}30\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}421\ E'_G - 0{,}079\ E'_B + 0{,}500\ E'_R$
5	Recommendation ITU-R BT.470-2 System B, G $E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$
6	SMPTE 170M $E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$
7	SMPTE 240M (1987) $E'_Y = 0{,}701\ E'_G + 0{,}087\ E'_B + 0{,}212\ E'_R$ $E'_{PB} = -0{,}384\ E'_G + 0{,}500\ E'_B - 0{,}116\ E'_R$ $E'_{PR} = -0{,}445\ E'_G - 0{,}055\ E'_B + 0{,}500\ E'_R$
8-255	reserved

In the case that sequence_display_extension() is not present in the bitstream or colour_description is zero the matrix coefficients are assumed to be those corresponding to matrix_coefficients having the value 1.

display_horizontal_size: See display_vertical_size.

display_vertical_size: display_horizontal_size and display_vertical_size together define a rectangle which may be considered as the “intended display’s” active region. If this rectangle is smaller than the encoded frame size then the display process may be expected to display only a portion of the encoded frame. Conversely if the display

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

rectangle is larger than the encoded frame size then the display process may be expected to display the reconstructed frames on a portion of the display device rather than on the whole display device.

display_horizontal_size shall be in the same units as horizontal_size (samples of the encoded frames).

display_vertical_size shall be in the same units as vertical_size (lines of the encoded frames).

display_horizontal_size and display_vertical_size do not affect the decoding process but may be used by the display process that is not standardised in this specification.

6.3.13.2.4 Quant matrix extension

Each quantisation matrix has a default set of values. When a video_object_start_code is decoded all matrices shall be reset to their default values. User defined matrices may be downloaded and this can occur in a quant_matrix_extension().

With 4:2:0 data only two matrices are used, one for intra blocks the other for non-intra blocks.

With 4:2:2 or 4:4:4 data four matrices are used. Both an intra and a non-intra matrix are provided for both luminance blocks and for chrominance blocks. Note however that it is possible to download the same user defined matrix into both the luminance and chrominance matrix at the same time.

The default matrix for intra blocks (both luminance and chrominance) is:

8	17	18	19	21	23	25	27
17	18	19	21	23	25	27	28
20	21	22	23	24	26	28	30
21	22	23	24	26	28	30	32
22	23	24	26	28	30	32	35
23	24	26	28	30	32	35	38
25	26	28	30	32	35	38	41
27	28	30	32	35	38	41	45

The default matrix for non-intra blocks (both luminance and chrominance) is:

16	17	18	19	20	21	22	23
17	18	19	20	21	22	23	24
18	19	20	21	22	23	24	25
19	20	21	22	23	24	26	27
20	21	22	23	25	26	27	28
21	22	23	24	26	27	28	30
22	23	24	26	27	28	30	31
23	24	25	27	28	30	31	33

load_intra_quantiser_matrix: This is a one-bit flag which is set to ‘1’ if intra_quantiser_matrix follows. If it is set to ‘0’ then there is no change in the values that shall be used.

intra_quantiser_matrix: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.4.2.1, replace the previous values. The first value shall always be 8 (values 1 to 7 and 9 to 255 are reserved). For all of the 8-bit unsigned integers, the value zero is forbidden. With 4:2:2 and 4:4:4 data the new values shall be used for both the luminance intra matrix and the chrominance intra matrix. However the chrominance intra matrix may subsequently be loaded with a different matrix except when rgb_component is set to '1'.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

load_non_intra_quantiser_matrix: This is a one-bit flag which is set to '1' if non_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used.

non_intra_quantiser_matrix: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.3.2.1, replace the previous values. For all the 8-bit unsigned integers, the value zero is forbidden. With 4:2:2 and 4:4:4 data the new values shall be used for both the luminance non-intra matrix and the chrominance non-intra matrix. However the chrominance non-intra matrix may subsequently be loaded with a different matrix.

load_chroma_intra_quantiser_matrix: This is a one-bit flag which is set to '1' if chroma_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used. If chroma_format is "4:2:0" or rgb_components is set to '1', this flag shall take the value '0'.

chroma_intra_quantiser_matrix: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.4.2.1, replace the previous values. The first value shall always be 8 (values 1 to 7 and 9 to 255 are reserved). For all of the 8-bit unsigned integers, the value zero is forbidden.

load_chroma_non_intra_quantiser_matrix: This is a one-bit flag which is set to '1' if chroma_non_intra_quantiser_matrix follows. If it is set to '0' then there is no change in the values that shall be used. If chroma_format is "4:2:0" or rgb_components is set to '1' this flag shall take the value '0'.

chroma_non_intra_quantiser_matrix: This is a list of sixty-four 8-bit unsigned integers. The new values, encoded in the default zigzag scanning order as described in 7.16.4.2.1, replace the previous values. For all the 8-bit unsigned integers, the value zero is forbidden.

load_intra_quantiser_matrix_grayscale[i]: This is a one-bit flag which is set to '1' if intra_quantiser_matrix_grayscale[i] follows. If it is set to '0' then there is no change in the values that shall be used.

intra_quantiser_matrix_grayscale[i]: This is a list of sixty-four 8-bit unsigned integers defining the grayscale intra alpha quantisation matrix to be used. The semantics and the default quantisation matrix are identical to those of intra_quantiser_matrix.

load_non_intra_quantiser_matrix_grayscale[i]: This is a one-bit flag which is set to '1' if non_intra_quantiser_matrix_grayscale[i] follows. If it is set to '0' then there is no change in the values that shall be used.

non_intra_quantiser_matrix_grayscale[i]: This is a list of sixty-four 8-bit unsigned integers defining the grayscale nonintra alpha quantisation matrix to be used. The semantics and the default quantisation matrix are identical to those of nonintra_quantiser_matrix.

6.3.13.2.5 Picture display extension

This specification does not define the display process. The information in this extension does not affect the decoding process and may be ignored by decoders that conform to this specification.

The picture display extension allows the position of the display rectangle whose size is specified in sequence_display_extension() to be moved on a picture-by-picture basis. One application for this is the implementation of pan-scan.

frame_centre_horizontal_offset: This is a 16-bit signed integer giving the horizontal offset in units of 1/16th sample. A positive value shall indicate that the centre of the reconstructed frame lies to the right of the centre of the display rectangle.

frame_centre_vertical_offset: This is a 16-bit signed integer giving the vertical offset in units of 1/16th sample. A positive value shall indicate that the centre of the reconstructed frame lies below the centre of the display rectangle.

The dimensions of the display rectangular region are defined in the sequence_display_extension(). The coordinates of the region within the coded picture are defined in the picture_display_extension().

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The centre of the reconstructed frame is the centre of the rectangle defined by video_object_layer_width and video_object_layer_height.

Since (in the case of an interlaced sequence) a coded VOP may relate to one, two or three decoded fields the picture_display_extension() may contain up to three offsets.

The number of frame centre offsets in the picture_display_extension() shall be defined as follows:

```
if ( progressive_sequence == 1) {
    if ( repeat_first_field == '1' ) {
        if ( top_field_first == '1' )
            number_of_frame_centre_offsets = 3
        else
            number_of_frame_centre_offsets = 2
    } else {
        number_of_frame_centre_offsets = 1
    }
} else {
    if (vop_structure == "field") {
        number_of_frame_centre_offsets = 1
    } else {
        if (repeat_first_field == '1' )
            number_of_frame_centre_offsets = 3
        else
            number_of_frame_centre_offsets = 2
    }
}
```

A picture_display_extension() shall not occur unless a sequence_display_extension() followed the StudioVideoObject().

In the case that a given picture does not have a picture_display_extension() then the most recently decoded frame centre offset shall be used. Note that each of the missing frame centre offsets have the same value (even if two or three frame centre offsets would have been contained in the picture_display_extension() had been present). Following a StudioVisualObject() the value zero shall be used for all frame centre offsets until a picture_display_extension() defines non-zero values.

Figure 6-22 illustrates the picture display parameters. As shown the frame centre offsets contained in the picture_display_extension() shall specify the position of the centre of the reconstructed frame from the centre of the display rectangle.

NOTES -

1 The display rectangle may also be larger than the reconstructed frame.

2 Even in a field VOP the frame_centre_vertical_offset still represents the offset of the centre of the frame in 1/16ths of a **frame** line (not a line in the field).

3 In the example of Figure 6-22 both frame_centre_horizontal_offset and frame_centre_vertical_offset have negative values.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

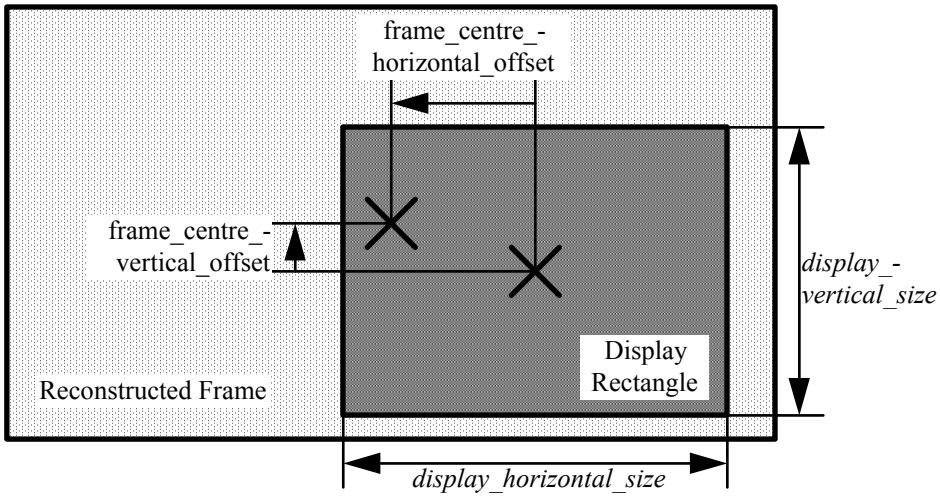


Figure 6-22 — Frame centre offset parameters

6.3.13.2.5.1 Pan-scan

The frame centre offsets may be used to implement pan-scan in which a rectangular region is defined which may be panned around the entire reconstructed frame.

By way of example only; this facility may be used to identify a 3/4 aspect ratio window in a 9/16 coded VOP format. This would allow a decoder to produce usable VOPs for a conventional definition television set from an encoded format intended for enhanced definition. The 3/4 aspect ratio region is intended to contain the “most interesting” region of the VOP.

The 3/4 region is defined by display_horizontal_size and display_vertical_size. The 9/16 frame size is defined by video_object_later_width and video_object_layer_height.

6.3.13.2.6 Copyright extension

copyright_flag: This is a one bit flag. When copyright_flag is set to ‘1’, it indicates that the source video material encoded in all the coded pictures following the copyright extension, in coding order, up to the next copyright extension or end of sequence code, is copyrighted. The copyright_identifier and copyright_number identify the copyrighted work. When copyright_flag is set to ‘0’, it does not indicate whether the source video material encoded in all the coded pictures following the copyright extension, in coding order, is copyrighted or not.

copyright_identifier: This is a 8-bit integer given by a Registration Authority as designated by ISO/IEC JTC1/SC29. Value zero indicates that this information is not available. The value of copyright_number shall be zero when copyright_identifier is equal to zero.

When copyright_flag is set to ‘0’, copyright_identifier has no meaning and shall have the value 0.

original_or_copy: This is a one bit flag. It is set to ‘1’ to indicate that the material is an original, and set to ‘0’ to indicate that it is a copy.

reserved: This is a 7-bit integer, reserved for future extension. It shall have the value zero.

copyright_number_1: This is a 20-bit integer, representing bits 44 to 63 of copyright_number.

copyright_number_2: This is a 22-bit integer, representing bits 22 to 43 of copyright_number.

copyright_number_3: This is a 22-bit integer. representing bits 0 to 21 of copyright_number.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

copyright_number: This is a 64-bit integer, derived from copyright_number_1, copyright_number_2, and copyright_number_3 as follows:

copyright_number **= (**copyright_number_1 << 44) + (copyright_number_2 << 22) + copyright_number_3.

The meaning of copyright_number is defined only when copyright_flag is set to '1'. In this case, the value of copyright_number identifies uniquely the copyrighted work marked by the copyrighted extension. The value 0 for copyright_number indicates that the identification number of the copyrighted work is not available.

When copyright_flag is set to '0', copyright_number has no meaning and shall have the value 0.

6.3.13.2.7 Camera parameters extension

camera_id – The number in camera_id identifies a camera.

height_of_image_device – This is a 22-bit unsigned integer which specifies the height of image device. Its value shall be measured to a resolution of 0,001 millimeter and having a range of zero to 4 194,303 mm.

focal_length – This is a 22-bit unsigned integer which specifies the focal length. Its value shall be measured to a resolution of 0,001 millimeter and having a range of zero to 4 194,303 mm.

f_number – This is a 22-bit unsigned integer which specifies the F-number. F-number is defined by (focal_length)/(effective aperture of lens). Its value shall be measured to a resolution of 0,001and having a range of zero to 4 194,303.

vertical_angle_of_view – This is a 22-bit unsigned integer which specifies the vertical angle of the field of view as determined between the top and bottom edges of the image device. Its value shall be measured to a resolution of 0,0001 degree and having a range of zero to 180 degrees.

camera_position_x_upper, camera_position_y_upper, camera_position_z_upper – These words constitute the 16 most significant bits of camera_position_x, camera_position_y and camera_position_z respectively.

camera_position_x_lower, camera_position_y_lower, camera_position_z_lower – These words constitute the 16 least significant bits of camera_position_x, camera_position_y and camera_position_z respectively.

camera_position_x, camera_position_y, camera_position_z – A set of these values specifies the position of the optical principal point of the camera in a user-specified world coordinate system. Each of these values shall be measured to a resolution of 0,001 millimeter and having a range of +2 147 483,647 mm to –2 147 483,648 mm. The camera_position_x is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_x_lower, the 16 most significant bits are defined in camera_position_x_upper. The camera_position_y is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_y_lower, the 16 most significant bits are defined in camera_position_y_upper. The camera_position_z is a 32-bit signed (two's complement) integer, the 16 least significant bits are defined in camera_position_z_lower, the 16 most significant bits are defined in camera_position_z_upper.

camera_direction_x, camera_direction_y, camera_direction_z – A set of these values specifies the direction of the camera. The direction of the camera is defined by using the vector from optical principal point to a point which is in front of the camera and is on the optical axis of the camera. Each of these values is a 22-bit signed (two's complement) integer and having a range of +2 097 151 to –2 097 152.

image_plane_vertical_x, image_plane_vertical_y, image_plane_vertical_z – A set of these values specifies the upper direction of the camera. The upper direction of the camera is defined by using the vector which is parallel to the side edge of the image device and is from bottom edge to top edge. Each of these values is a 22-bit signed (two's complement) integer and is having a range of +2 097 151 to –2 097 152.

Figure 6-23 explains these terms pictorially.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

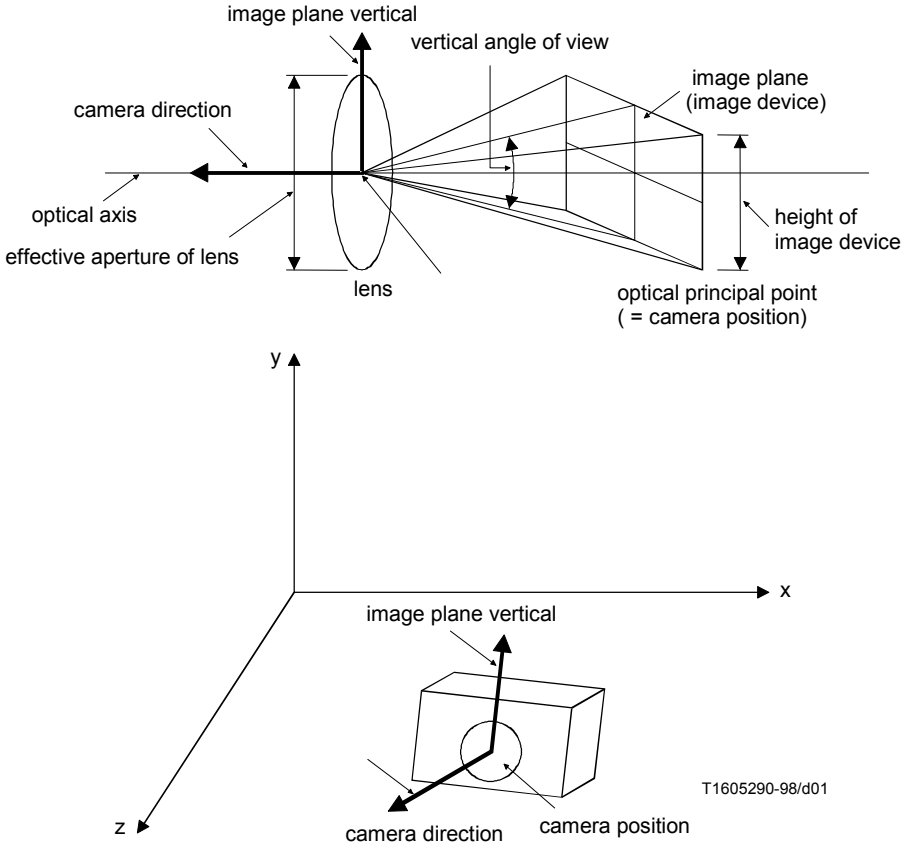


Figure 6-23 — Camera parameters

6.3.13.2.8 ITU-T extension

The use of this extension is defined in Annex A of ITU-T Recommendation H.320.

6.3.13.2.9 VLC code extension

In order to prevent start code emulation in StudioMacroblock() or StudioBlock(), a set of VLC codes that does not cause start code emulation for any combination of syntax elements shall be encoded.

load_vlc_code: This is a four bit integer indicating the presence of downloadable VLC codes for quantised AC coefficients. The value of load_intra_vlc_code and load_inter_vlc_code is derived from this code. Semantics are defined in the Table 6-80.

Table 6-80 — Meaning of load_vlc_code

load_vlc_code	Meaning
0000	forbidden
0001	load_intra_vlc_code = 1 load_inter_vlc_code = 0
0010	load_intra_vlc_code = 0 load_inter_vlc_code = 1
0011	load_intra_vlc_code = 1 load_inter_vlc_code = 1
0100 – 1111	reserved

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If load_intra_vlc_code is set to '1', the VLC codes defined in Table B.50 to Table 6-61 shall be replaced with the codes indicated by intra_vlc_length[j][i] and intra_vlc_code[j][i]. If it is set to '0', there is no change in Table B.50 to Table 6-61.

If load_inter_vlc_code is set to '1', the VLC codes defined in Table B.62 to Table B.73 shall be replaced with the codes indicated by inter_vlc_length[j][i] and inter_vlc_code[j][i]. If it is set to '0', there is no change in Table B.62 to Table B.73.

intra_vlc_length[j][i]: This is a four bit unsigned integer indicating the length of the VLC code of the entry No. i in the Table No. j, which is defined in Table B.50 to Table 6-61. The actual length is intra_vlc_length[j][i]+1. The length of each VLC code in each table shall be limited to less than 17. The length of the VLC code of entry No.21 in each table shall also be limited less than 14.

intra_vlc_code[j][i]: This is a 16 bit unsigned integer indicating the VLC code of the entry No. i in the Table No. j, which is defined in Table B.50 to Table 6-61. The most significant bits of the length indicated by intra_vlc_length[j][i] is the actual VLC code, and the remaining bits shall be filled with '1'.

inter_vlc_length[j][i]: This is a four bit unsigned integer indicating the length of the VLC code of the entry No. i in the Table No. j, which is defined in Table B.62 to Table B.73. The actual length is inter_vlc_length[j][i]+1. The length of each VLC code in each table shall be limited to less than 17. The length of the VLC code of entry No.21 in each table shall also be limited less than 14.

inter_vlc_code[j][i]: This is a 16 bit unsigned integer indicating the VLC code of the entry No. i in the Table No. j, which is defined in Table B.62 to Table B.73. The most significant bits of the length indicated by intra_vlc_length[j][i] is the actual VLC code, and the remaining bits shall be filled with '1'.

6.3.13.3 Studio Video Object Layer

video_object_layer_start_code: The video_object_layer_start_code is a string of 32 bits. The first 28 bits are '0000 0000 0000 0000 0000 0001 0010' in binary and the last 4-bits represent one of the values in the range of '0000' to '1111' in binary. The video_object_layer_start_code identifies the beginning of a video object layer header.

random_accessible_vol: This flag may be set to "1" to indicate that every VOP in this VOL is individually decodable. If all of the VOPs in this VOL are intra-coded VOPs and some more conditions are satisfied then random_accessible_vol may be set to "1". The flag random_accessible_vol is not used by the decoding process. random_accessible_vol is intended to aid random access or editing capability. This shall be set to "0" if any of the VOPs in the VOL are non-intra coded or certain other conditions are not fulfilled.

video_object_type_indication: Constrains the following bitstream to use tools from the indicated object type only, e.g. Simple Object or Core Object, as shown in Table 6-81.

Table 6-81 — FLC table for video_object_type indication

Video Object Type	Code
Reserved	00000000
Simple Object Type	00000001
Simple Scalable Object Type	00000010
Core Object Type	00000011
Main Object Type	00000100
N-bit Object Type	00000101
Basic Anim. 2D Texture	00000110
Anim. 2D Mesh	00000111
Simple Face	00001000

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Still Scalable Texture	00001001
Advanced Real Time Simple	00001010
Core Scalable	00001011
Advanced Coding Efficiency	00001100
Advanced Scalable Texture	00001101
Simple Studio Object Type	00001110
Core Studio Object Type	00001111
Reserved	00010000 - 11111111

video_object_layer_verid: This is a 4-bit code which identifies the version number of the video object layer. Its meaning is defined in Table 6-82. If both visual_object_verid and video_object_layer_verid exist, the semantics of video_object_layer_verid supersedes the other.

Table 6-82 — Meaning of video_object_layer_verid

video_object_layer_verid	Meaning
0000	reserved
0001	object type listed in Table 9-1
0010	object type listed in Table 9-2
0011	reserved
0100	object type listed in Table 9-3
0101 – 1111	reserved

video_object_layer_shape: This is a 2-bit integer defined in Table 6-83. It identifies the shape type of a video object layer.

Table 6-83 — Video Object Layer shape type

Shape format	Meaning
00	rectangular
01	binary
10	binary only
11	grayscale

video_object_layer_shape_extension: This is a 4-bit integer defined in Table 6-84. It identifies the number (up to 3) and type of auxiliary components that can be used, including the grayscale shape (ALPHA) component. Only a limited number of types and combinations are defined in Table 6-83. More applications are possible by selection of the USER DEFINED type. If the video_object_layer_shape does not indicate grayscale shape, this code shall be set to '1111'.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-84 — Semantic meaning of video_object_layer_shape_extension

video_object_layer_ shape_extension	aux_comp_type[0]	aux_comp_type[1]	aux_comp_type[2]	aux_comp_ count
0000	ALPHA	NO	NO	1
0001	DISPARITY	NO	NO	1
0010	ALPHA	DISPARITY	NO	2
0011	DISPARITY	DISPARITY	NO	2
0100	ALPHA	DISPARITY	DISPARITY	3
0101	DEPTH	NO	NO	1
0110	ALPHA	DEPTH	NO	2
0111	TEXTURE	NO	NO	1
1000	USER DEFINED	NO	NO	1
1001	USER DEFINED	USER DEFINED	NO	2
1010	USER DEFINED	USER DEFINED	USER DEFINED	3
1011	ALPHA	USER DEFINED	NO	2
1100	ALPHA	USER DEFINED	USER DEFINED	3
1101-1110	t.b.d.	t.b.d.	t.b.d.	t.b.d.
1111	NO	NO	NO	

progressive_sequence: When set to ‘1’ the coded video sequence contains only progressive frame-VOPs. When progressive_sequence is set to ‘0’ the coded video sequence may contain both frame-VOPs and field-VOPs, and frame-VOP may be progressive or interlaced frames.

rgb_components: this is a one bit flag indicating if the components of the sequence are the RGB format or not. If it is set to '1', the components are RGB and the component representing green color is coded in the same way as the luminance component and the blue and red components are coded in the same way as the chrominance components. That is, the green component shall be treated as the Y component, the blue component shall be as the Cb component, and the red component shall be as the Cr component. This flag does not affect the decoding process except the restriction on the value of chroma_format and parameters related to the quantiser matrices.

chroma_format: This is a two bit integer indicating the chrominance format as defined in the Table 6-85. If rgb_components is set to '1', this shall be set to '11'.

Table 6-85 — Meaning of chroma_format

chroma_format	Meaning
00	reserved
01	4:2:0
10	4:2:2
11	4:4:4

bits_per_pixel: This field specifies the video data precision in bits per pixel. It may take different values for different video object layers within a single video object. A value of 12 in this field would indicate 12 bits per pixel. This field may take values between 4 and 12. The same number of bits per pixel is used in the luminance and two chrominance planes.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

video_object_layer_width: The video_object_layer_width is a 14-bit unsigned integer representing the width of the displayable part of the luminance component of VOPs in pixel units. The width of the encoded luminance component of VOPs in macroblocks is (video_object_layer_width+15)/16. The displayable part is left-aligned in the encoded VOPs.

video_object_layer_height: The video_object_layer_height is a 14-bit unsigned integer representing the height of the displayable part of the luminance component of the frame in lines.

In the case that progressive_sequence is '1' the height of the encoded luminance component of frames in macroblocks, mb_height, is (video_object_layer_height+15)/16.

In the case that progressive_sequence is '0' the height of the encoded luminance component of frame VOPs in macroblocks, mb_height, is 2*((video_object_layer_height + 31)/32). The height of the encoded luminance component of field VOPs in macroblocks, mb_height, is ((video_object_layer_height + 31)/32).

The displayable part is top-aligned in the encoded VOPs.

aspect_ratio_info: This is a four-bit integer which defines the value of pixel aspect ratio. Table 6-86 shows the meaning of the code. If aspect_ratio_info indicates extended PAR, pixel_aspect_ratio is represented by par_width and par_height. The par_width and par_height shall be relatively prime.

Table 6-86 — Meaning of pixel aspect ratio

aspect_ratio_info	pixel aspect ratios
0000	Forbidden
0001	1:1 (Square)
0010	12:11 (625-type for 4:3 picture)
0011	10:11 (525-type for 4:3 picture)
0100	16:11 (625-type stretched for 16:9 picture)
0101	40:33 (525-type stretched for 16:9 picture)
0110-1110	Reserved
1111	extended PAR

par_width: This is an 8-bit unsigned integer which indicates the horizontal size of pixel aspect ratio. A zero value is forbidden.

par_height: This is an 8-bit unsigned integer which indicates the vertical size of pixel aspect ratio. A zero value is forbidden.

frame_rate_code: This is a four-bit integer used to define frame_rate as shown in Table 6-87.

If progressive_sequence is '1' the period between two successive frames at the output of the decoding process is the reciprocal of the frame_rate. If progressive_sequence is '0' the period between two successive fields at the output of the decoding process is half of the reciprocal of the frame_rate.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-87 — frame_rate

frame_rate_code	frame_rate
0000	forbidden
0001	24 000÷1001 (23,976…)
0010	24
0011	25
0100	30 000÷1001 (29,97…)
0101	30
0110	50
0111	60 000÷1001 (59,94…)
1000	60
…	reserved
1111	reserved

The VBV constraint is defined in Annex D.

first_half_bit_rate, latter_half_bit_rate: The bit rate is a 30-bit unsigned integer which specifies the bitrate of the bitstream measured in units of 400 bits/second, rounded upwards. The value zero is forbidden. This value is divided to two parts. The most significant bits are in first_half_bit_rate (15 bits) and the least significant bits are in latter_half_bit_rate (15 bits). The marker_bit is inserted between the first_half_bit_rate and the latter_half_bit_rate in order to avoid the start code emulation. The instantaneous video object layer channel bit rate seen by the encoder is denoted by $R_{vol}(t)$ in bits per second. If the bit_rate (i.e. first_half_bit_rate and latter_half_bit_rate) field in the VOL header is present, it defines a peak rate (in units of 400 bits per second; a value of 0 is forbidden) such that $R_{vol}(t) <= 400 \times bit_rate$ Note that $R_{vol}(t)$ counts only visual syntax for the current elementary stream (also see Annex D).

first_half_vbv_buffer_size, latter_half_vbv_buffer_size: vbv_buffer_size is an 18-bit unsigned integer. This value is divided into two parts. The most significant bits are in first_half_vbv_buffer_size (15 bits) and the least significant bits are in latter_half_vbv_buffer_size (3 bits), The VBV buffer size is specified in units of 16384 bits. The value 0 for vbv_buffer_size is forbidden. Define B = 16384 × vbv_buffer_size to be the VBV buffer size in bits.

first_half_vbv_occupancy, latter_half_vbv_occupancy: The vbv_occupancy is a 26-bit unsigned integer. This value is divided to two parts. The most significant bits are in first_half_vbv_occupancy (11 bits) and the least significant bits are in latter_half_vbv_occupancy (15 bits). The marker_bit is inserted between the first_vbv_buffer_size and the latter_half_vbv_buffer_size in order to avoid the start code emulation. The value of this integer is the VBV occupancy in 64-bit units just before the removal of the first VOP following the VOL header. The purpose for the quantity is to provide the initial condition for VBV buffer fullness.

low_delay: This is a one-bit flag which when set to ‘1’ indicates the VOL contains no B-VOPs.

mpeg2_stream: This is a one bit flag which when set to ‘1’ indicates if the decoded bitstream is transcoded from one which conforms to ISO/IEC 13818-2:2000 according to information specified in ANNEX Q. This flag affects the inverse quantisation process as defined in 7.16.4.3 in order to decode the bitstream accurately.

alpha_bits_per_pixel[i]: This field specifies the video data precision of alpha planes for grayscale alpha or auxiliary component i=0,1,2 in bits per pixel. It may take different values for different video object layers within a single video object. A value of 12 in this field would indicate 12 bits per pixel. This field may take values between 4 and 12.

minimum_alpha_level[i]: T his is a 12-bit unsigned integer which specifies the level for complete transparency of alpha signals for grayscale alpha or auxiliary component i=0,1,2.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

maximum_alpha_level[i]: This is a 12-bit unsigned integer which specifies the level for complete opacity of alpha signals_for grayscale alpha or auxiliary component i=0,1,2. maximum_alpha_level shall be grater than minimum_alpha_level.

composition_method: This is a one bit flag which indicates which blending method is to be applied to the video object in the compositor. When set to '0', cross-fading shall be used. When set to '1', additive mixing shall be used. See subclause 7.16.6.3.5.

linear_composition: This is a one bit flag which indicates the type of signal used by the compositing process. When set to '0', the video signal in the format from which it was produced by the video decoder is used. When set to '1', linear signals are used. See subclause 7.16.6.3.5.

sprite_enable: This is a one-bit flag which when set to '1' indicates the presence of sprites.

sprite_width: This is a 20-bit unsigned integer which identifies the horizontal dimension of the sprite.

sprite_height: This is a 20-bit unsigned integer which identifies the vertical dimension of the sprite.

sprite_left_coordinate: This is a 20-bit signed integer which defines the left-edge of the sprite. The value of sprite_left_coordinate shall be divisible by two.

sprite_top_coordinate: This is a 20-bit signed integer which defines the top edge of the sprite. The value of sprite_left_coordinate shall be divisible by two.

no_of_sprite_warping_points: This is a 6-bit unsigned integer which represents the number of points used in sprite warping. When its value is 0 and when sprite_enable is set to '1', warping is identity (stationary sprite) and no coordinates need to be coded. When its value is 4, a perspective transform is used. When its value is 1,2 or 3, an affine transform is used. Further, the case of value 1 is separated as a special case from that of values 2 or 3. Table 6-88 shows the various choices.

Table 6-88 — Number of point and implied warping function

Number of points	warping function
0	Stationary
1	Translation
2,3	Affine
4	Perspective

sprite_warping_accuracy: This is a 2-bit code which indicates the quantisation accuracy of motion vectors used in the warping process for sprites. Table 6-89 shows the meaning of various codewords.

Table 6-89 — Meaning of sprite warping accuracy codewords

code	sprite_warping_accuracy
00	1/2 pixel
01	1/4 pixel
10	1/8 pixel
11	1/16 pixel

sprite_brightness_change: This is a one-bit flag which when set to '1' indicates a change in brightness during sprite warping ; alternatively, a value of '0' means no change in brightness.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sprite_defocusing: This is a one-bit flag which when set to `1` indicates a change in defocusing during sprite warping; alternatively, a value of `0` means no change in defocusing.

sprite_lens_distortion: This is a one-bit flag which when set to `1` indicates that the lens distortion exists during sprite warping; alternatively, a value of `0` means there is no lens distortion.

6.3.13.4 Group of Studio Video Object Plane

group_vop_start_code: The group_vop_start_code is the bit string '000001B3' in hexadecimal. It identifies the beginning of a Group of Studio VOP header.

closed_gov: This is a one-bit flag which indicates the nature of the predictions used in the first consecutive B-VOPs (if any) immediately following the first coded I-VOP after the group of studio VOP header. The closed_gov is set to '1' to indicate that these B-VOPs have been encoded using only backward prediction or intra coding. This bit is provided for use during any editing which occurs after encoding. If the previous VOPs have been removed by editing, broken_link may be set to '1' so that a decoder may avoid displaying these B-VOPs following the first I-VOP following the group of studio VOP header. However if the closed_gov bit is set to '1', then the editor may choose not to set the broken_link bit as these B-VOPs can be correctly decoded.

broken_link: This is a one-bit flag which shall be set to '0' during encoding. It is set to '1' to indicate that the first consecutive B-VOPs (if any) immediately following the first coded I-frame following the group of studio VOP header may not be correctly decoded because the reference frame which is used for prediction is not available (because of the action of editing). A decoder may use this flag to avoid displaying frames that cannot be correctly decoded.

6.3.13.4.1 Time Code SMPTE12M

Time_code[63...0] (64bit): Time-code format defined in Annex P shall be transparent with SMPTE 12M.

6.3.13.5 Studio Video Object Plane

vop_start_code: This is the bit string '000001B6' in hexadecimal. It marks the start of a video object plane.

temporal_reference: The temporal_reference is a 10-bit unsigned integer associated with each coded VOP.

The following simple specification applies only when low_delay is equal to zero.

When a coded frame is in the form of two field VOPs, the temporal_reference associated with each VOP shall be the same (it is called the temporal_reference of the coded frame). The temporal_reference of each coded frame shall increment by one modulo 1024 when examined in display order at the output of the decoding process, except when a group of studio VOP header occurs. Among the frames coded after a group of studio VOP header, the temporal_reference of the coded frame that is displayed first shall be set to zero.

The following more general specification applies when low_delay is equal to zero or one.

If VOP A is not a big VOP, i.e., the VBV buffer is only examined once before the coded VOP A is removed from the VBV buffer, and if N is the temporal_reference of VOP A, then the temporal_reference of VOP B immediately following VOP A in display order is equal to :

- 0 if there is a group of studio VOP header present between VOP A and VOP B (in coded order).
- (N+1) % 1024 if VOP B is a frame VOP or is the first of a pair of field VOPs.
- N if VOP B is the second field of a pair of field VOPs.

When low_delay is equal to one, there may be situations where the VBV buffer shall be re-examined several times before removing a coded VOP (referred to as a big VOP) from the VBV buffer.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If VOP A is a big VOP and if K is the number of times that the VBV buffer is re-examined as defined in Annex D (K>0), if N is the temporal_reference of VOP A, then the temporal_reference of VOP B immediately following VOP A in display order is equal to :

- K % 1024 if there is a group of studio VOP header present between VOP A and VOP B (in coded order).
- (N+K+1) % 1024 if VOP B is a frame VOP or is the first field of a pair of field VOPs.
- (N+K) % 1024 if VOP B is the second field of a pair of field VOPs.

  NOTE - If the big VOP is the first field of a frame coded with field VOPs, then the temporal_reference of the two field VOPs of that coded frame are not identical.

vop_structure: This is a 2-bit integer defined in the Table 6-90.

Table 6-90 — Meaning of vop_structure

vop_structure	Meaning
00	reserved
01	Top Field
10	Bottom Field
11	Frame picture

When a frame is encoded in the form of two field VOPs both fields must be of the same vop_coding_type, except where the first encoded field is an I-VOP in which case the second may be either an I-VOP or a P-VOP.

The first encoded field of a frame may be a top-field or a bottom field, and the next field must be of opposite parity.

When a frame is encoded in the form of two field VOPs the following syntax elements may be set independently in each field VOP:

- vop_fcode[0][0], vop_fcode[0][1]
- vop_fcode[1][0], vop_fcode[1][1]
- dct_precision, alpha_dct_precision[i]
- intra_dc_precision, q_scale_type, alpha_intra_dc_precision[i], alpha_q_scale_type[i]
- alternate_scan
  - temporal_reference

vop_coding_type: The vop_coding_type identifies whether a VOP is an intra-coded VOP (I), predictive-coded VOP (P), bidirectionally predictive-coded VOP (B) or sprite VOP (S). The meaning of vop_coding_type is defined in Table 6-91.

Table 6-91 — Meaning of vop_coding_type

vop_coding_type	coding method
00	intra-coded (I)
01	predictive-coded (P)
10	bidirectionally-predictive-coded (B)
11	sprite (S)

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

vop_coded: This is a 1-bit flag which when set to '0' indicates that no subsequent data exists for the VOP. In this case, the following decoding rule applies: For an arbitrarily shaped VO (i.e. when the shape type of the VO is either 'binary' or 'binary only'), the alpha plane of the reconstructed VOP shall be completely transparent. For a rectangular VO (i.e. when the shape type of the VO is 'rectangular'), the corresponding rectangular alpha plane of the VOP, having the same size as its luminance component, shall be completely transparent. If there is no alpha plane being used in the decoding and composition process of a rectangular VO, the reconstructed VOP is filled with the respective content of the immediately preceding VOP for which vop_coded!=0.

vop_width: This is a 14-bit unsigned integer which specifies the horizontal size, in pixel units, of the rectangle that includes the VOP. The width of the encoded luminance component of VOP in macroblocks is (vop_width+15)/16. The rectangle part is left-aligned in the encoded VOP. A zero value is forbidden.

vop_height: This is a 14-bit unsigned integer which specifies the vertical size, in pixel units, of the rectangle that includes the VOP. The height of the encoded luminance component of VOP in macroblocks is (vop_height+15)/16. The rectangle part is top-aligned in the encoded VOP. A zero value is forbidden.

vop_horizontal_mc_spatial_ref: This is a 14-bit signed integer which specifies, in pixel units, the horizontal position of the top left of the rectangle defined by horizontal size of vop_width. The value of vop_horizontal_mc_spatial_ref shall be divisible by two. This is used for decoding and for picture composition.

vop_vertical_mc_spatial_ref: This is a 14-bit signed integer which specifies, in pixel units, the vertical position of the top left of the rectangle defined by vertical size of vop_height. The value of vop_vertical_mc_spatial_ref shall be divisible by two for progressive and divisible by four for interlaced motion compensation. This is used for decoding and for picture composition.

top_field_first: The meaning of this element depends upon vop_structure, progressive_sequence and repeat_first_field.

If progressive_sequence is equal to '0', this flag indicates what field of a reconstructed frame is output first by the decoding process:

In a field VOP top_field_first shall have the value '0', and the only field output by the decoding process is the decoded field VOP.

In a frame VOP top_field_first being set to '1' indicates that the top field of the reconstructed frame is the first field output by the decoding process. top_field_first being set to '0' indicates that the bottom field of the reconstructed frame is the first field output by decoding process

If progressive_sequence is equal to '1', this flag, combined with repeat_first_field, indicates how many times (one, two or three) the reconstructed frame is output by the decoding process.

If repeat_first_field is set to 0, top_field_first shall be set to '0'. In this case the output of the decoding process corresponding to this reconstructed frame consists of one progressive frame.

If top_field_first is set to 0 and repeat_first_field is set to '1', the output of the decoding process corresponding to this reconstructed frame consists of two identical progressive frames.

If top_field_first is set to 1 and repeat_first_field is set to '1', the output of the decoding process corresponding to this reconstructed frame consists of three identical progressive frames.

repeat_first_field: This flag is applicable only in a frame VOP, in a field VOP it shall be set to zero and does not affect the decoding process.

If progressive_sequence is equal to 0 and progressive_frame is equal to 0, repeat_first_field shall be zero, and the output of the decoding process corresponding to this reconstructed frame consists of two fields.

If progressive_sequence is equal to 0 and progressive_frame is equal to 1:

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If this flag is set to 0, the output of the decoding process corresponding to this reconstructed frame consists of two fields. The first field (top or bottom field as identified by top_field_first) is followed by the other field.

If it is set to 1, the output of the decoding process corresponding to this reconstructed frame consists of three fields. The first field (top or bottom field as identified by top_field_first) is followed by the other field, then the first field is repeated.

If progressive_sequence is equal to 1:

If this flag is set to 0, the output of the decoding process corresponding to this reconstructed frame consists of one frame.

If it is set to 1, the output of the decoding process corresponding to this reconstructed frame consists of two or three frames, depending on the value of top_field_first.

progressive_frame: If progressive_frame is set to 0 it indicates that the two fields of the frame are interlaced fields in which an interval of time of the field period exists between (corresponding spatial samples) of the two fields. In this case the following restriction applies:

- repeat_first_field shall be zero (two field duration).

If progressive_frame is set to 1 it indicates that the two fields (of the frame) are actually from the same time instant as one another. In this case a number of restrictions to other parameters and flags in the bitstream apply:

- vop_structure shall be “Frame”

- if progressive_sequence is equal to one, frame_pred_frame_dct shall be 1

intra_predictors_reset: This flag indicates whether the macroblock to be decoded shall be decoded independently of any parameters of the previous macroblocks of the current VOP even if the macroblock is not located at the start of a slice. If this flag is set to 1, each predictor, such as for intra dc coefficient, is reset in the same way as done at the start of a sliece. Otherwise, predictions are used between neighbouring macroblocks in the same slice.

alternate_scan: This flag affects the decoding of transform coefficient data as described in 7.16.4.2.

frame_pred_frame_dct: If this flag is set to ‘1’ then only frame-DCT and frame prediction are used. In a field VOP it shall be ‘0’. frame_pred_frame_dct shall be ‘1’ if progressive_sequence is ‘1’. This flag affects the syntax of the bitstream.

dct_precision: This is a 2-bit integer defining the value of base_quantiser scale according to Table 6-92. This flag affects the inverse quantisation process as defined in 7.16.4.3.

Table 6-92 — DCT precision

dct_precision	base_quantiser scale
00	1.000
01	0.500
10	0.250
11	0.125

In case of mpeg2_stream = 1, dct_precision shall be set to ‘00’.

intra_dc_precision: This is a 2-bit integer defined in the Table 6-93.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-93 — Intra DC precision

intra_dc_precision	Precision (bits)
00	bits_per_pixel
01	bits_per_pixel+1
10	bits_per_pixel+2
11	bits_per_pixel+3

The inverse quantisation process for the Intra DC coefficients is modified by this parameter as explained in 7.16.3.3.1.

q_scale_type: This flag affects the inverse quantisation process as described in 7.16.4.3.2.2.

vop_fcode[s][t]: A 4 bit unsigned integer taking values 1 through 9, or 15. The value zero is forbidden and the values 10 through 14 are reserved. It is used in the decoding of motion vectors, see 7.16.7.4.1.

In an I-VOP vop_fcode[s][t] is not used (since motion vectors are not used) and shall take the value 15 (all ones).

Similarly, in an I-VOP or a P-VOP vop_fcode[1][t] is not used in the decoding process (since it refers to backwards motion vectors) and shall take the value 15 (all ones).

See Table 7-38 for the meaning of the indices; s and t.

dead_zone_disable: This is a one bit flag which affect the inverse quantisation process as described in 7.16.4.2.3.

alpha_dct_precision[i]: This is a 2-bit integer defined in the Table 6-94. This flag affects the inverse quantisation process of the alpha macroblock in the same way as dct_precision does for that of the luminance macroblock.

Table 6-94 — alpha base quantiser scale

alpha_dct_precision	alpha_base_quantiser scale
00	1.000
01	0.500
10	0.250
11	0.125

alpha_intra_dc_precision[i]: This is a 2-bit integer defined in the Table 6-95.

Table 6-95 — Alpha Intra DC precision

alpha_intra_dc_precision	Precision (bits)
00	bits_per_pixel
01	bits_per_pixel+1
10	bits_per_pixel+2
11	bits_per_pixel+3

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The inverse quantisation process for the Intra DC coefficients of alpha channel is modified by this parameter in the same way as the luminance component.

alpha_q_scale_type[i]: This flag affects the inverse quantisation process for alpha channel in the same way as the luminance components.

vop_sprite_brightness_change: This is a one-bit flag which when set to '1' indicates a change in brightness during sprite warping, alternatively, a value of '0' means no change in brightness.

composite_display_flag: This flag is set to 1 to indicate that the following fields are of use when the input pictures have been coded as (analogue) composite video prior to encoding into a bitstream that complies with this specification. If it is set to 0 then these parameters do not occur in the bitstream.

The information relates to the picture that immediately follows the extension. In the case that this picture is a frame picture the information relates to the first field of that frame. The equivalent information for the second field may be derived (there is no way to represent it in the bitstream).

NOTES

1 The various syntactic elements that are included in the bitstream if composite_display_flag is '1' are not used in the decoding process.

2 repeat_first_field will cause a composite video field to be repeated out of the 4-field or 8-field sequence. It is recommended that repeat_first_field and composite_display_flag are not both set simultaneously.

v_axis: A 1-bit integer used only when the bitstream represents a signal that had previously been encoded according to PAL systems. v_axis is set to 1 on a positive sign, v_axis is set to 0 otherwise.

field_sequence: A 3-bit integer which defines the number of the field in the eight field sequence used in PAL systems or the four field sequence used in NTSC systems as defined in the Table 6-96.

Table 6-96 — Definition of field_sequence

field sequence	frame	field
000	1	1
001	1	2
010	2	3
011	2	4
100	3	5
101	3	6
110	4	7
111	4	8

sub_carrier: This is a 1-bit integer. When set to zero it means the sub-carrier/line frequency relationship is correct. When set to 1 the relationship is not correct.

burst_amplitude: This is a 7-bit integer defining the burst amplitude (for PAL and NTSC only). The amplitude of the sub-carrier burst is quantised as a Recommendation ITU-R BT.601 luminance signal, with the MSB omitted.

sub_carrier_phase: This is an 8-bit integer defining the phase of the reference sub-carrier at the field-synchronisation datum with respect, to field start as defined in Recommendation ITU-R BT.470. See Table 6-97.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-97 — Definition of sub_carrier_phase

sub_carrier_phase	Phase
0	([360°÷256] * 0)
1	([360°÷256] * 1)
...	...
255	([360°÷256] * 255)

extra_bit_picture: A bit indicates the presence of the following extra information. If extra_bit_picture is set to '1', extra_information_picture will follow it. If it is set to '0', there are no data following it. extra_bit_picture shall be set to '0', the value '1' is reserved for possible future extensions defined by ITU-T|ISO/IEC.

extra_information_picture: Reserved. A decoder conforming to this specification that encounters extra_information_picture in a bitstream shall ignore it (i.e. remove from the bitstream and discard). A bitstream conforming to this specification shall not contain this syntax element.

6.3.13.6 Studio Sprite coding

warping_mv_code(dmv): The codeword for each differential motion vector consists of a VLC indicating the length of the dmv code (dmv_length) and a FLC, dmv_code-, with dmv_length bits. The codewords are listed in Table B.88.

brightness_change_factor (): The codeword for brightness_change_factor consists of a variable length code denoting brightness_change_factor_size and a fixed length code, brightness_change_factor, of brightness_change_factor_size bits (sign bit included). The codewords are listed in Table B.89.

defocusing_control_parameter: This specifies the defocusing control parameter for luminance and chrominance sprite images by the following defocusing process.

The defocusing is performed by a filtering process. The applied filter $P_s(x, y)$ is shown in Figure 6-24.

$$P_s(x,y) = \begin{cases} 1/4a^2 & |x| \le a, |y| \le a \\ 0 & else \end{cases}$$

The value "a" is used as the defocusing control parameter.

The defocusing control parameter "a" for chrominance signals is divided by two in vertical direction or horizontal direction according to the chroma format.

- 4:4:4 format"a" is not divided.

- 4:2:2 format"a" is divided by two in horizontal direction.

- 4:2:0 format"a" is divided by two both in horizontal direction and in vertical direction.

In case the filter taps lie outside the image, outer pixels shall be obtained by the padding process as defined in subclause 7.16.5.9.1.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

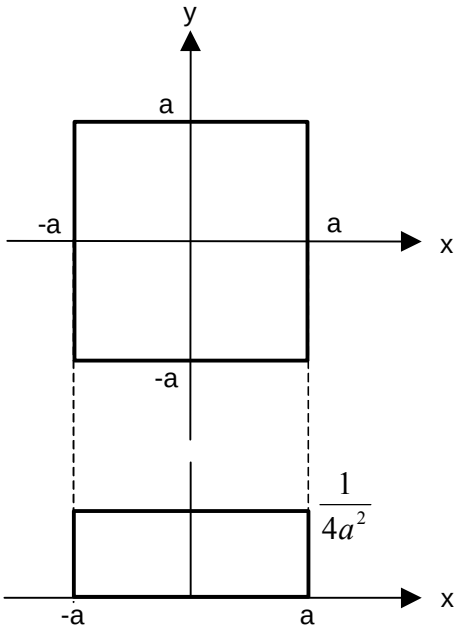


Figure 6-24 — Defocusing filter

The length of this value varies according to the sprite warping accuracy. Table 6-98 shows the length of the defocusing control parameter.

Table 6-98 — Length of defocusing control parameter

sprite warping accuracy	length of defocusing control parameter	defocusing control parameter
½ pixel	9 bits	0-511
¼ pixel	10 bits	0-1023
1/8 pixel	11 bits	0-2047
1/16 pixel	12 bits	0-4095

lens_distortion_parameter(): The codeword for lens_distortion_parameter consists of the following 4 syntax elements. **lens_distortion_parameter_1** and **lens_distortion_parameter_2** are 16-bit signed (two's complement) integer and **lens_center_horizontal** and **lens_center_vertical** are 14-bit unsigned integer. These parameters represent camera lens distortion of the following equation.

$$x' = u(\tilde{x} - c_x) + c_x$$
$$y' = u(\tilde{y} - c_y) + c_y$$

with

$$u = 1 + R(k_1 \cdot \tfrac{1}{4W^3} + Rk_2 \cdot \tfrac{1}{2W^5})$$
$$R = (\tilde{x} - c_x)^2 + s^2(\tilde{y} - c_y)^2$$

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

where

$(\tilde{x}, \tilde{y})$ is the position of a distorted point

(x', y') is the position of no radial lens distortion point

(c_x, c_y) is the lens center position in the image coordinates

(**lens_center_horizontal, lens_center_vertical**)

k_1 is the first lens distortion coefficient

lens_distortion_parameter_1

k_2 is the second lens distortion coefficient

lens_distortion_parameter_2

W is the maximum image size

the larger one of **video_object_layer_width** or **video_object_layer_height** in StudioVideoObjectLayer()

s is the aspect ratio of each pixel

aspect_ratio_info in StudioVideoObjectLayer()

Note: These syntax elements provide no normative decoding procedure.

6.3.13.7 Studio Slice

slice_start_code: This is the bit string '000001B7' in hexadecimal. It marks the start of a slice.

macroblock_number: This is a variable length code with length between 1 and 14 bits. It identifies the macroblock number within a VOP. The number of the top-left macroblock in a VOP shall be zero. The macroblock number increases from left to right and from top to bottom. The actual length of the code depends on the total number of macroblocks in the VOP calculated according to Table 6-99, the code itself is a binary representation of the macroblock number.

Table 6-99 — Length of macroblock_number code

length of macroblock_number code	((vop_width+15)/16) * ((vop_height+15)/16)
1	1-2
2	3-4
3	5-8
4	9-16
5	17-32
6	33-64
7	65-128
8	129-256
9	257-512
10	513-1024

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11	1025-2048
12	2049-4096
13	4097-8192
14	8193-16384

quantiser_scale_code: A 5 bit unsigned integer in the range 1 to 31. The decoder shall use this value until another quantiser_scale_code is encountered either in StudioSlice() or StudioMacroblock(). The value zero is forbidden.

alpha_quantiser_scale_code[i]: A 5 bit unsigned integer in the range 1 to 31. The decoder shall use this value until another alpha_quantiser_scale_code[i] is encountered either in StudioSlice() or StudioMacroblock(). The value zero is forbidden.

slice_extension_flag: This flag shall be set to '1' to indicate the presence of intra_slice, slice_VOP_enable and slice_VOP_id in the bitstream.

intra_slice: This flag shall be set to '0' if any of the macroblocks in the slice are non-intra macroblocks. If all of the macroblocks are intra macroblocks then intra_slice may be set to '1'. intra_slice may be omitted from the bitstream (by setting intra_slice_flag to '0') in which case it shall be assumed to have the value zero.

intra_slice is not used by the decoding process. intra_slice is intended to aid a DSM application in performing FF/FR.

slice_VOP_id_enable: This flag controls the semantics of slice_VOP_id. If slice_VOP_id_enable is set to "0", slice_VOP_id is not used by this specification and shall have the value zero. If slice_VOP_id_enable is set to "1", slice_VOP_id may have a value different from zero.

slice_VOP_id_enable must have the same value in all the slices of a VOP. slice_VOP_id_enable may be omitted from the bitstream (by setting slice_extension_flag to '0') in which case it shall be assumed to have the value zero.

slice_VOP_id_enable is not used by the decoding process.

slice_VOP_id: This is a 6 bit integer. If slice_VOP_id_enable is set to "0", slice_VOP_id is not used by this specification and shall have the value zero. If slice_VOP_id_enable is set to "1", slice_VOP_id is application defined and may have any value, with the constraint that slice_VOP_id shall have the same value in all the slices of a VOP.

slice_VOP_id is not used by the decoding process. slice_VOP_id is intended to aid recovery on severe bursts of errors for certain types of applications. For example the application may increment slice_VOP_id with each transmitted VOP, so that in case of severe burst error, when several slices are lost, the decoder can know if the slice following the burst error belongs to the current VOP or to another VOP, which may be the case if at least a VOP header has been lost.

extra_bit_slice: This flag indicates the presence of the following extra information. If extra_bit_slice is set to '1', extra_information_slice will follow it. If it is set to '0', there are no data following it. extra_bit_slice shall be set to '0', the value '1' is reserved for possible future extensions defined by ITU-T|ISO/IEC.

extra_information_slice: Reserved. A decoder conforming to this specification that encounters extra_information_slice in a bitstream shall ignore it (i.e. remove from the bitstream and discard). A bitstream conforming to this specification shall not contain this syntax element.

6.3.13.8 Studio Macroblock

not_coded: This is a 1-bit flag which signals if a macroblock is coded or not. When set to'1' it indicates that a macroblock is not coded and no further data is included in the bitstream for this macroblock (with the exception of alpha data that may be present). The decoder shall treat this macroblock as 'inter' with motion vector equal to zero

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

and no DCT coefficient data for P-VOPs. When set to '0' it indicates that the macroblock is coded and its data is included in the bitstream.

compression_mode: This is a flag which is set to '0' to indicate that the texture component of a current macroblock shall be coded following the DPCM block syntax. Otherwise, the macroblock is coded as DCT data.

dpcm_scan_order: This is a flag that indicates the scanning order of blocks for DPCM coding. If set to value '0' the block is scanned from top line to bottom line, and from left to right. If set to value '1' the block is scanned from bottom line to top line, and from right to left.

coda_i[j]: This is a one-bit flag which is set to "1" to indicate that all the values in the grayscale alpha macroblock are equal to maximum_alpha_level[j] (AlphaOpaqueValue[j]). When set to "0", this flag indicates that one or more 8x8 blocks are coded according to alpha_pattern_code[j].

alpha_compression_mode[j]: This is a flag which is set to '0' to indicate that the alpha component 'j' of a current macroblock shall be coded following the DPCM block syntax. Otherwise, the macroblock is coded as DCT data.

alpha_macroblock_quant[j]: This is set to 1 to indicate that alpha_quantiser_scale_code[j] is present in the bitstream.

alpha_quantiser_scale_code[j]: A 5 bit unsigned integer in the range 1 to 31. The decoder shall use this value until another alpha_quantiser_scale_code[j] is encountered either in StudioSlice() or StudioMacroblock(). The value zero is forbidden.

alpha_dpcm_scan_order[j]: This is a flag that indicates the scanning order of a alpha block for DPCM coding. If set to value '0' the block is scanned from top line to bottom line. If set to value '1' the block is scanned from bottom line to top line.

coda_pb[j]: This is a VLC indicating the coding status for P alpha macroblocks. The semantics are given in the table below (Table 6-100). When this VLC indicates that the alpha macroblock is all opaque, this means that all values are set to maximum_alpha_level[j] (AlphaOpaqueValue[j]).

Table 6-100 — coda_pb[j] codes and corresponding values

coda_pb[j]	Meaning
1	alpha residue all zero
01	alpha macroblock all opaque
00	alpha residue coded

cbpa[j]: This is the coded block pattern for an inter macroblock of grayscale alpha texture data. This VLC is defined in Table B.83 - Table B.86. cbpa is followed by the alpha block data which is coded in the same way as texture block data. Note that grayscale alpha blocks in an inter macroblock with alpha all equal to 0 (alpha residual all zero) are not included in the bitstream.

alpha_pattern_code[i]: The value of this internal flag is set to 1 if the alpha block with the index value i (i=0,…,3) includes one or more DCT coefficients that are decoded through the same process as the luminance components. Otherwise, the value of this flag shall be set to 0. For an alpha intra macroblock, alpha_pattern_code[i] shall be set to 1, and for an alpha inter macroblock alpha_pattern_code[i] shall be derived from cbpa[j].

6.3.13.8.1 Studio MB Binary Shape Coding

bab_type: This is a variable length code between 1 and 6 bits. It indicates the coding mode used for the bab. There are five bab_types as depicted in Table 6-101. The VLC tables used depend on the decoding context i.e. the bab_types of blocks already received. For I-VOPs, the context-switched VLC table of Table B.75 is used. For P-VOPs, the context-switched table of Table B.76 is used.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-101 — List of bab_types and usage

bab_type	Semantic	Used in
0	MVs==0 && No Update	P- VOPs
1	MVs!=0 && No Update	P- VOPs
2	transparent	All VOP types
3	Opaque	All VOP types
4	HHC	All VOP types

The bab_type determines what other information fields will be present for the bab shape. No further shape information is present if the bab_type = 0, 2 or 3. Opaque means that all pixels of the bab are part of the object. Transparent means that none of the bab pixels belong to the object. HHC means the Hierarchical Huffman decoding will be required to reconstruct the pixel of the bab. No_update means that motion compensation is used to copy the bab from the previous VOP's binary alpha map.

mvs_x: This is a variable-length code between 1 and 18 bits. It represents the horizontal element of the motion vector difference for the bab. The motion vector is in full integer precision. The VLC table is shown in Table B.77.

mvs_y: This is a variable-length code between 1 and 18 bits. It represents the vertical element of the motion vector difference for the bab. The motion vector is in full integer precision. If mvs_x is '0', then the VLC table of Table B.78, otherwise the VLC table of Table B.77 is used.

inferior_symbol_macroblock: This is a 1-bit flag indicating less frequent pixel between the opaque and transparent pixel in the macroblock. If this flag is set to "0", the transparent symbol is inferior and the opaque symbol is superior. If this flag is set to "1", the opaque symbol is inferior and the transparent is superior.

inferior_symbol_block: This is a 1-bit flag indicating less frequent pixel between the opaque and transparent pixel in the block. If this flag is set to "0", the transparent symbol is inferior and the opaque symbol is superior. If this flag is set to "1", the opaque symbol is inferior and the transparent is superior.

cbbp: This is a variable-length code between 3 to 7 bits. It indicates the existence of inferior symbol in each block. The VLC table is shown in Table B.79.

scan_direction: This is a 1-bit flag indicating whether the block is divided into horizontal rows or vertical columns. If this flag is set to "0", the block is divided into horizontal rows. If this flag is set to "1", the block is divided into vertical columns.

backward_load_flag: This is a 1-bit flag indicating whether the order of the line bit pattern is changed. If this flag is set to "0", the order of the line bit pattern isn't changed. If this flag is set to "1", the order of the line bit pattern is turned right side left in scan_direction=="0" or upside down in scan_direction=='1'.

clp: This is a variable-length code between 2 to 24 bits. It indicates the existence of an inferior symbol in each pixel line. The VLC table is shown in Table B.80.

lbp: This is a variable-length code between 2 to 23 bits. It indicates the bit pattern of an inferior symbol in the pixel line. The VLC table is shown in Table B.81.

6.3.13.8.2 Studio Macroblock modes

macroblock_type: Variable length coded indicator which indicates the method of coding and content of the macroblock according to the Table B.42 and Table B.43, selected by vop_coding_type.

macroblock_quant: Derived from macroblock_type according to the Table B.42 and Table B.43. This is set to 1 to indicate that quantiser_scale_code is present in the bitstream.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

macroblock_motion_forward: Derived from macroblock_type according to the Table B.42 and Table B.43. This flag affects the bitstream syntax and is used by the decoding process.

macroblock_pattern: Derived from macroblock_type according to the Table B.42 and Table B.43. This is set to 1 to indicate that coded_block_pattern() is present in the bitstream.

macroblock_intra: Derived from macroblock_type according to the Table B.42 and Table B.43. This flag affects the bitstream syntax and is used by the decoding process.

frame_motion_type: This is a two bit code indicating the macroblock prediction type, defined in Table 6-102.

If frame_pred_frame_dct is equal to 1 then frame_motion_type is omitted from the bitstream. In this case motion vector decoding and prediction formation shall be performed as if frame_motion_type had indicated “Frame-based prediction”.

Table 6-102 — Meaning of frame_motion_type

code	prediction type	motion_vector _count	mv_format
00	reserved		
01	Field-based	2	field
10	Frame-based	1	frame
11	reserved		

field_motion_type: This is a two bit code indicating the macroblock prediction type, defined in Table 6-103.

Table 6-103 — Meaning of field_motion_type

code	prediction type	motion_vector _count	mv_format
00	reserved		
01	Field-based	1	field
10	16x8 MC	2	field
11	reserved		

dct_type: This is a flag indicating whether the macroblock is frame DCT coded or field DCT coded. If this is set to ‘1’, the macroblock is field DCT coded; otherwise, the macroblock is frame DCT coded. Boundary blocks are always coded in frame-based mode.

In the case that **dct_type** is not present in the bitstream then the value of dct_type (used in the remainder of the decoding process) shall be derived as shown in Table 6-104.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-104 — Value of dct_type if dct_type is not in the bitstream.

Condition	dct_type
vop_structure == “field”	unused because there is no frame/field distinction in a field vop.
frame_pred_frame_dct == 1	0 (“frame”)
!(macroblock_intra || macroblock_pattern)	unused - macroblock is not coded
macroblock is skipped	unused - macroblock is not coded
compression_mode == "DPCM"	0("frame") - This is used for alpha macroblock.
vop_structure == "frame" && frame_pred_frame_dct == 0 && macroblock_pattern == 0 && macroblock_intra == 0	0("frame") - This is used for alpha macroblock.

6.3.13.8.3 Motion vectors

motion_vector_count is derived from field_motion_type or frame_motion_type as indicated in the Table 6-102 and Table 6-103.

mv_format is derived from field_motion_type or frame_motion_type as indicated in the Table 6-102 and Table 6-103. mv_format indicates if the motion vector is a field-motion vector or a frame-motion vector. mv_format is used in the syntax of the motion vectors and in the process of motion vector prediction.

motion_vertical_field_select[r][s]: This flag indicates which reference field shall be used to form the prediction. If motion_vertical_field_select[r][s] is zero then the top reference field shall be used, if it is one then the bottom reference field shall be used. (See 7.16.5.4.1 for the meaning of the indices; r and s.)

6.3.13.8.4 Motion vector

motion_code[r][s][t]: This is a variable length code, as defined in 7.16.5.4.1, which is used in motion vector decoding as described in 7.16.5.4.1. (See Table 7-38 for the meaning of the indices; r, s and t.)

motion_residual[r][s][t] : This is an integer which is used in motion vector decoding as described in 7.16.5.4.1. (See Table 7-38 for the meaning of the indices; r, s and t.). The number of bits in the bitstream for motion_residual[r][s][t], r_size, is derived from vop_fcode[s][t] as follows;

$$r_size = vop_fcode[s][t] - 1$$

NOTE - The number of bits for both motion_residual[0][s][t] and motion_residual[1][s][t] is denoted by vop_fcode[s][t].

6.3.13.8.5 Coded block pattern

coded_block_pattern_420: A variable length code that is used to derive the variable cbp according to Table B.82.

coded_block_pattern_1 --

coded_block_pattern_2: For 4:2:2 and 4:4:4 data the coded block pattern is extended by the addition of either a two bit or six bit fixed length code, coded_block_pattern_1 or coded_block_pattern_2. Then the pattern_code[i] is derived using the following:

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
for (i=0; i<12; i++) {
    if (macroblock_intra)
        pattern_code[i] = 1;
    else
        pattern_code[i] = 0;
}
if (macroblock_pattern) {
    for (i=0; i<6; i++)
        if ( cbp & (1<<(5-i)) ) pattern_code[i] = 1;
    if (chroma_format == “4:2:2”)
        for (i=6; i<8; i++)
            if ( coded_block_pattern_1 & (1<<(7-i)) ) pattern_code[i] = 1;
    if (chroma_format == “4:4:4”)
        for (i=6; i<12; i++)
            if ( coded_block_pattern_2 & (1<<(11-i)) ) pattern_code[i] = 1;
}
```

If pattern_code[i] is equal to 1, i=0 to (block_count-1), then the block number i defined in Figure 6-8, Figure 6-9 and Figure 6-10 is contained in this macroblock.

The number “block_count” which determines the number of blocks in the macroblock is derived from the chrominance format as shown in Table 6-105.

Table 6-105 — block_count as a function of chroma_format

chroma_format	block_count
4:2:0	6
4:2:2	8
4:4:4	12

If the block number i is transparent, transparent_block(i)=1, the velue of pattern_code[i] does not affect the decoding process.

If the block number i is transparent, transparent_block(i)=1, the velue of pattern_code[i] does not affect the decoding process.

6.3.13.9 Studio Block

dct_dc_size_luminance: This is a variable length code as defined in Table B.44 that is used to derive the value of the differential DC coefficients of luminance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

dct_dc_differential: This is a variable length code as defined in Table B.46 that is used to derive the value of the differential DC coefficients in blocks in intra macroblocks. After identifying the category of the DC coefficient in size from dct_dc_size_luminance or dct_dc_size_chrominance, this value denotes which actual difference in that category occurred.

dct_dc_size_chrominance: This is a variable length code as defined in Table B.45 that is used to derive the value of the differential DC coefficients of chrominance values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

6.3.13.9.1 Studio Alpha Block

alpha_pattern_code[i]: The value of this internal flag is set to 1 if the alpha block with the index value i indicates one or more DCT coefficients that are decoded in the same way as the luminance component. Otherwise the value of this flag is set to 0.

dct_dc_size_alpha: This is a variable length code as defined in Table B.44 that is used to derive the value of the differential DC coefficients of alpha values in blocks in intra macroblocks. This value categorizes the coefficients according to their size.

The other semantics of StudioAlphaBlock() are described in clause 7.

6.3.13.10 Studio DPCM Block

block_mean: This is an unsigned integer that indicates the average value of pixels within a DPCM coded block. This mean value is also used for efficient prediction of the sign of residuals in DPCM coding. The value 0 is forbidden.

rice_parameter: This is an unsigned integer that indicates the length of the **rice_suffix_code** field. The value 0 is forbidden. The value 15 shall be interpreted as 0. The values 12 - 14 are also forbidden.

rice_prefix_code: This is a variable length code that represents the most significant bits of a DPCM residual. This code may also represent an escape sequence in which case a DPCM residual is coded by **dpcm_residual** instead of a combination of **rice_prefix_code** and **rice_suffix_code**.

dpcm_residual: This is an unsigned integer that indicates the value of a DPCM residual.

rice_suffix_code: This is an unsigned integer that represents the least significant bits of a DPCM residual.

Table 6-106 — Variable length codes for rice_prefix_code

Variable length code	rice_prefix_code
1	0
01	1
001	2
0001	3
0000 1	4
0000 01	5
0000 001	6
0000 0001	7
0000 0000 1	8
0000 0000 01	9
0000 0000 001	10
0000 0000 0001	escape

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.14 FGS Video Object

6.3.14.1 FGS Video Object Plane

fgs_vop_start_code: This is the bit string '000001B9' in hexadecimal. It marks the start of an fgs vop.

fgs_vop_coding_type: The fgs_vop_coding_type identifies whether an fgs vop is an FGS coding type (I), predictive-coded FGS coding type (P), or bidirectionally predictive-coded FGS coding type (B). The meaning of fgs_vop_coding_type is defined in Table 6-107.

Table 6-107 — Meaning of fgs_vop_coding_type

fgs_vop_coding_type	coding method
00	FGS (I)
01	predictive-coded FGS (P)
10	bidirectionally-predictive-coded FGS (B)
11	reserved

fgs_vop_max_level_y: This is an unsigned integer which specifies the number of vop-bps in which the Y component has at least one non-zero block-bp in the first one of them.

fgs_vop_max_level_u: This is an unsigned integer which specifies the number of vop-bps in which the U component has at least one non-zero block-bp in the first one of them.

fgs_vop_max_level_v: This is an unsigned integer which specifies the number of vop-bps in which the V component has at least one non-zero block-bp in the first one of them.

fgs_vop_number_of_vop_bp_coded: This is an unsigned integer which specifies the number of vop-bps coded into the bitstream.

fgs_vop_mc_bit_plane_used: This parameter is reserved for future use. The value of this parameter shall be 0.

fgs_vop_selective_enhancement_enable: This flag shall be 1 when selective enhancement is enabled and it shall be 0 otherwise.

fgs_ref_select_code: This is a 2-bit unsigned integer which specifies prediction reference choices for "P" and "B" fgs vops with respect to decoded reference layer identified by ref_layer_id. The meaning of allowed values is specified in Table 6-108 and Table 6-109.

Table 6-108 — Prediction reference choices for "P" fgs vops

fgs_ref_select_code	forward prediction reference
00	Reserved
01	Most recently VOP in display order belonging to the reference layer.
10	Next VOP in display order belonging to the reference layer.
11	Reserved

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-109 — Prediction reference choices for "B" fgs vops

fgs_ref_select_code	forward temporal reference	backward temporal reference
00	Reserved	
01	Reserved	
10	Reserved	
11	Most recently VOP in display order belonging to the reference layer.	Next VOP in display order belonging to the reference layer.

fgs_bp_start_code: This is a string of 32 bits. The first 27 bits are '0000 0000 0000 0000 0000 0001 010' in binary and the last 5 bits represent one of the values in the range of '00000' to '11111' in binary. The fgs_bp_start_code marks a new vop-bp.

fgs_vop_bp_id: This is given by the last 5 bits of the fgs_bp_start_code. The fgs_vop_bp_id uniquely identifies a vop-bp. The value of fgs_vop_bp_id is "0" for the most significant vop-bp and increments by 1 for each successively lower vop-bp.

fgs_resync_marker: This is a binary string of 22 zeros followed by a one '000 0000 0000 0000 0000 0001'. It is only present when fgs_resync_marker_disable flag is set to '0'.

fgs_macroblock_number: This is a variable length code with length between 1 and 14 bits. It identifies the fgs macroblock number of the following fgs macroblock. The actual length of the code depends on the total number of fgs macroblocks in a vop-bp, calculated according to Table 6-110. The code itself is a binary representation of the fgs macroblock number.

Table 6-110 — Length of fgs_macroblock_number code

length of fgs_macroblock_number code	((video_object_layer_width+15)/16) * ((video_object_layer_height+15)/16)
1	1-2
2	3-4
3	5-8
4	9-16
5	17-32
6	33-64
7	65-128
8	129-256
9	257-512
10	513-1024
11	1025-2048
12	2049-4096
13	4097-8192
14	8193-16384

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.14.2 FGS Motion Macroblock

fgs_not_coded: This is a 1-bit flag that signals if an FGS motion macroblock is coded or not in a "P" fgs vop. When set to'1' it indicates that an FGS motion macroblock is not coded and no further data is included in the bitstream for this FGS motion macroblock; decoder shall treat this FGS motion macroblock as motion vector equal to zero. When set to '0' it indicates that the FGS motion macroblock is coded and its data is included in the bitstream.

fgs_p_mb_type: This is a 1-bit flag that signals the type of motion vector coding in "P" fgs vop. When set to "0", it indicates that the one-motion-vector mode is used in this FGS motion macroblock. When set to "1", it indicates that the four-motion-vector mode is used in this FGS motion macroblock.

fgs_modb: This is a 1-bit flag that signals if an FGS motion macroblock is coded or not in a "B" fgs vop. When set to'1' it indicates that an FGS motion macroblock is not coded and no further data is included in the bitstream for this FGS motion macroblock; decoder shall treat this FGS motion macroblock as motion vector equal to zero. When set to '0' it indicates that the FGS motion macroblock is coded and its data is included in the bitstream.

fgs_b_mb_type: This variable length code is present only in coded FGS motion macroblocks of "B" fgs vops. The codes for fgs_b_mb_type are shown in Table 6-111.

Table 6-111 — VLC table for fgs_b_mb_type

Code	fgs_b_mb_type
1	direct
01	interpolate
001	backward
0001	forward

6.3.14.3 FGS Motion Interlaced Information

fgs_field_prediction: This is a 1-bit flag indicating whether the FGS motion macroblock is field predicted or frame predicted. This flag is set to '1' when the FGS motion macroblock is predicted using field motion vectors. If it is set to '0' then frame prediction (16x16 or 8x8) will be used. This flag is only present in the bitstream if the interlaced flag is set to "1", and either the fgs_p_mb_type is "0" in a "P" fgs vop or the FGS motion macroblock is in a "B" fgs vop.

fgs_forward_top_field_reference: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the top field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not backward predicted.

fgs_forward_bottom_field_reference: This is a 1-bit flag which indicates the reference field for the forward motion compensation of the bottom field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not backward predicted.

fgs_backward_top_field_reference: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the top field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not forward predicted.

fgs_backward_bottom_field_reference: This is a 1-bit flag which indicates the reference field for the backward motion compensation of the bottom field. When this flag is set to '0', the top field is used as the reference field. If it is set to '1' then the bottom field will be used as the reference field. This flag is only present in the bitstream if the fgs_field_prediction flag is set to "1" and the FGS motion macroblock is not forward predicted.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.3.14.4 FGS Motion Vector

Identical to subclause 6.3.6.2.

6.3.14.5 FGS Macroblock

fgs_cbp: This is a variable length code with length between 1 and 9 bits. It specifies the coded bit pattern of fgs_msb_not_reached in the fgs macroblock in the first two vop-bps.

For the first vop-bp with Y, U, and V components, Table 6-112 summarizes the code for fgs_cbp: (note: x denotes either 0 or 1)

Table 6-112 — Coding table for fgs_cbp in the first vop-bp with Y, U, and V components

Code	Pattern of fgs_msb_not_reached yyyy,uv	Meaning
1	1111,11	6 all-zero block-bps
01xxxx	xxxx,11	U and V block-bps are all-zero block-bps, but at least one of the 4 Y block-bps is not an all-zero block-bp
00xxxx01	xxxx,00	both U and V block-bps are not all-zero block-bps
00xxxx10	xxxx,01	U block-bp is not an all-zero block-bp and V block-bp is an all-zero block-bp
00xxxx11	xxxx,10	U block-bp is an all-zero block-bp and V block-bp is not an all-zero block-bp

Within a vop-bp, one of the three color components may be absent. If one of the U and V components is absent, the above coding scheme is followed except that the absent color component is not coded. Table 6-113 specifies this case:

Table 6-113 — Coding table for fgs_cbp in the first vop-bp when one of the U and V components is absent

Code	Pattern of fgs_msb_not_reached yyyy,u/v	Meaning
1	1111,1	5 all-zero block-bps
0xxxxx (000001 –011111)	xxxx,x (0000,0 – 1111,0)	not all 5 blocks are all-zero block-bps

To avoid start code emulation, the 5-bit pattern of fgs_msb_not_reached forms a 5-bit integer and the integer plus 1 in binary is ‘xxxxx’ in the code.

If both U and V components are absent in vop-bp, Table 6-114 specifies the code:

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 6-114 — Coding table for fgs_cbp in the first vop-bp when both U and V components are absent

Code	Pattern of fgs_msb_not_reached yyyy	Meaning
1	1111	4 all-zero block-bps
0100 0101 0110 0111	0111 1011 1101 1110	3 all-zero block-bps
001000 001001 001010 001011 001100 001101	0011 0101 0110 1001 1010 1100	2 all-zero block-bps
000100 000101 000110 000111	1000 0100 0010 0001	1 all-zero block-bp
00001	0000	no all-zero block-bp

In the following unlikely cases:

- Y component is absent,
- the Y and U components are absent,
- the Y and V components are absent,

the pattern of fgs_msb_not_reached is put into the bitstream without VLC coding.

For the second vop-bp, the pattern of fgs_msb_not_reached is put into the bitstream without VLC coding except for the fgs macroblocks that satisfy the following conditions:

- Y, U, and V components of the second vop-bp are all present;
- The pattern of fgs_msb_not_reached for the fgs macroblock in the first vop-bp with the same fgs macroblock number is all 1's.

The coding method for this case is specified in Table 6-115:

Table 6-115 — Coding table for fgs_cbp in the second vop-bp

Code	Pattern of fgs_msb_not_reached yyyy,uv	Meaning
011	1111,11	6 all-zero block-bps
010	0000,11	4 Y block-bps are not all-zero block-bps and both U and V block-bps are all-zero block-bps
1xxxx	xxxx,11	at least one Y block-bp is all-zero block-bp but not all 4 Y block-bps are all-zero block-bps, and both U and V block-bps are all-zero block-bps

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10000	0000,00	none of the 6 block-bps are all-zero block-bps
11111	1111,00	4 Y block-bps are all-zero block-bps and both U and V block-bps are not all-zero block-bps
00100	0000,01	4 Y block-bps and the U block-bp are not all-zero block-bps, but the V block-bp is an all-zero block-bp
00101	0000,10	4 Y block-bps and the V block-bp are not all-zero block-bps, but the U block-bp is an all-zero block-bp
00110	1111,01	4 Y block-bps and the V block-bp are all-zero block-bps, but the U block-bp is not an all-zero block-bp
00111	1111,10	4 Y block-bps and the U block-bp are all-zero block-bps, but the V block-bp is not an all-zero block-bp
000xxxxxx	xxxx,xx	All other cases

fgs_shifted_bits: This is a variable length code with length between 1 and 5 bits as shown in Table 6-116. When selective enhancement is used, all DCT coefficients in an fgs macroblock are left-shifted by fgs_shifted_bits.

Table 6-116 — VLC table of fgs_shifted_bits

Number of bits	VLC code
0	1
1	01
2	001
3	0001
4	00001

fgs_dct_type: This is a 1-bit flag indicating whether this fgs macroblock is frame DCT coded or field DCT coded. If this flag is set to “1”, the fgs macroblock is field DCT coded; otherwise, the fgs macroblock is frame DCT coded. This flag is only present in the bitstream if the interlaced flag is set to “1” in the fgs vop header. This flag is only present once in the first non-zero fgs macroblock with the same fgs macroblock number.

6.3.14.6 FGS Block

fgs_msb_not_reached: This flag is 1 if the most significant block-bp in an 8x8 DCT block is not reached yet. In other words, this indicates an all-zero block-bp before the most significant block-bp. This flag is 0 otherwise.

fgs_run_eop_code: The VLC code for a (RUN,EOP) symbol as described in Clause 7.

fgs_sign_bit: This bit indicates the sign of the DCT coefficient with value 0 indicating positive and value 1 indicating negative.

7 The visual decoding process

This clause specifies the decoding process that the decoder shall perform to recover visual data from the coded bitstream. As shown in Figure 7-1, the visual decoding process includes several decoding processes such as shape-motion-texture decoding, still texture decoding, mesh decoding, and face decoding processes. After decoding the coded bitstream, it is then sent to the compositor to integrate various visual objects.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

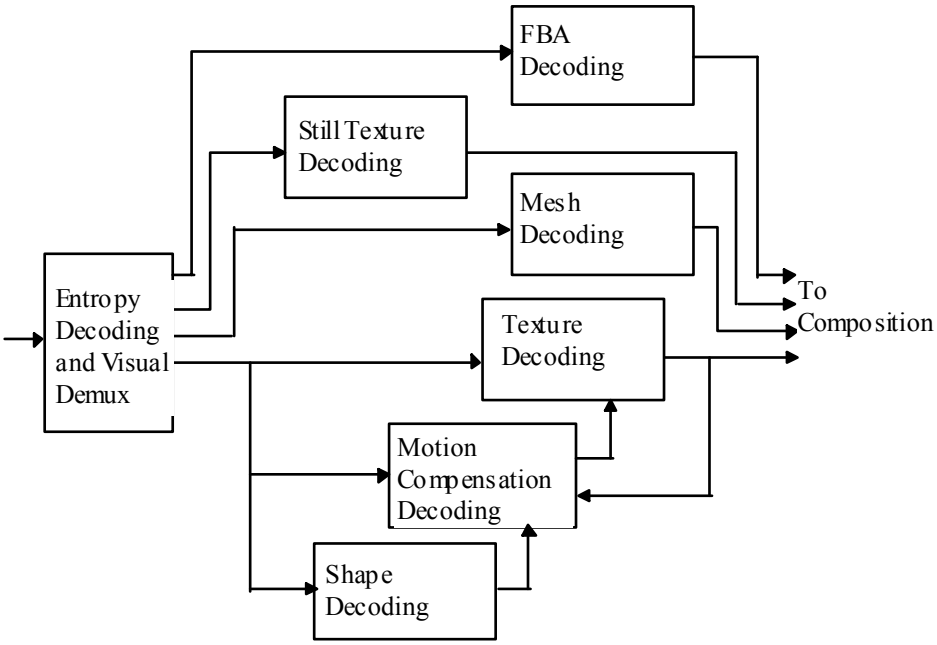


Figure 7-1 — A high level view of basic visual decoding; specialized decoding such as scalable, sprite and error resilient decoding are not shown

In subclauses 7.1 through 7.9 the VOP decoding process is specified in which shape, motion, texture decoding processes are the major contents. The still texture object decoding is described in subclauses 7.10. Subclause 7.11 includes the mesh decoding process, and subclause 7.12 features the face object decoding process. The output of the decoding process is explained in subclause 7.13. The video object decoding for the studio profile is specified in subclause 7.16.

7.1 Video decoding process

This subclause specifies the decoding process that a decoder shall perform to recover VOP data from the coded video bitstream.

With the exception of the Inverse Discrete Cosine Transform (IDCT) the decoding process is defined such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here, by definition, complies with this part of ISO/IEC 14496.

The IDCT is defined statistically such that different implementations for this function are allowed. The IDCT specification is given in Annex A.

Figure 7-2 is a diagram of the Video Decoding Process without any scalability feature. The diagram is simplified for clarity. The same decoding scheme is applied when decoding all the VOPs of a given session

NOTE Throughout this part of ISO/IEC 14496 two dimensional arrays are represented as *name*[q][p] where ‘q’ is the index in the vertical dimension and ‘p’ the index in the horizontal dimension.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

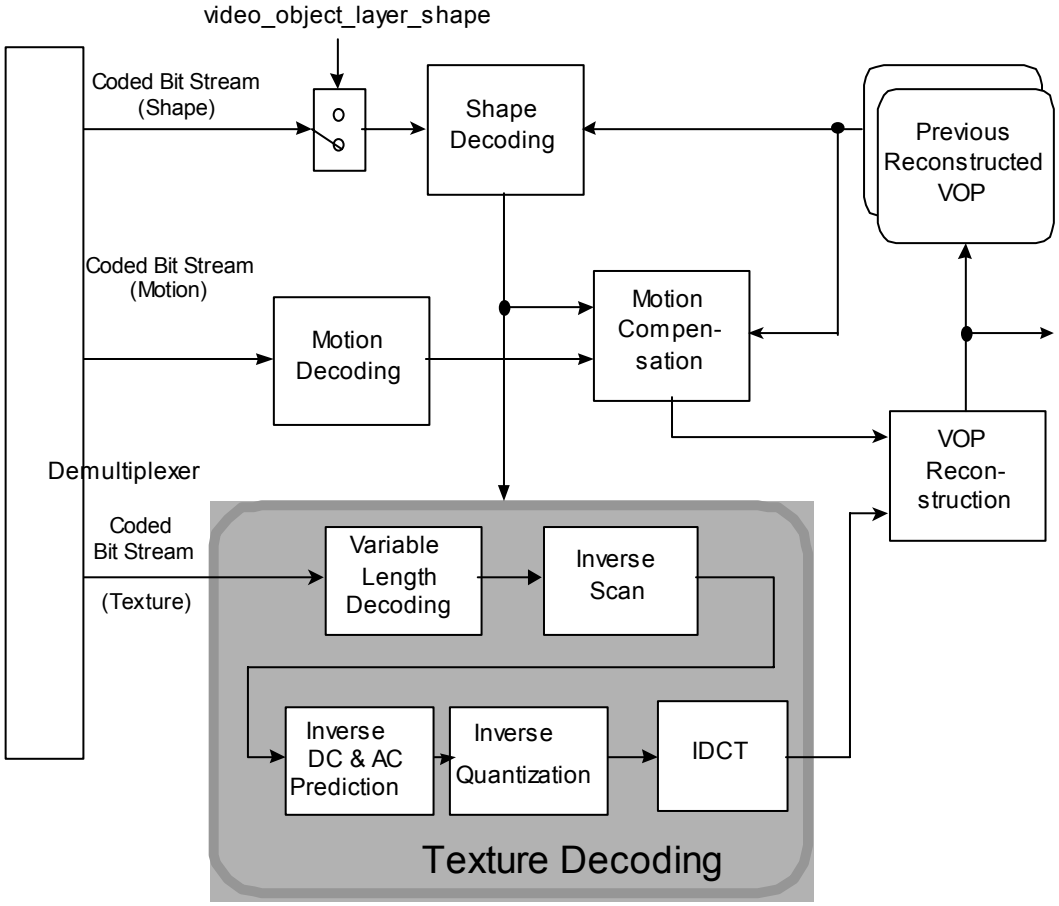


Figure 7-2 — Simplified Video Decoding Process

The decoder is mainly composed of three parts: shape decoder, motion decoder and texture decoder. The reconstructed VOP is obtained by combining the decoded shape, texture and motion information.

7.2 Higher syntactic structures

The various parameters and flags in the bitstream for VideoObjectLayer(), Group_of_VideoObjectPlane(), VideoObjectPlane(), video_plane_with_short_header(), macroblock() and block(), as well as other syntactic structures related to them shall be interpreted as specified earlier. Many of these parameters and flags affect the decoding process. Once all the macroblocks in a given VOP have been processed, the entire VOP will have been reconstructed. In case the bitstream being decoded contains B-VOPs, reordering of VOPs may be needed as specified in subclause 6.1.3.7.

7.3 VOP reconstruction

The luminance and chrominance values of a VOP from the decoded texture and motion information are reconstructed as follows:

1. In case of intra macroblocks, the luminance and chrominance values f[y][x] from the decoded texture data form the luminance and chrominance values of the VOP: d[y][x] = f[y][x].

2. In case of inter macroblocks, first the prediction values p[y][x] are calculated using the decoded motion vector information and the texture information of the respective reference VOPs. Then, the decoded texture data f[y][x] is added to the prediction values, resulting in the final luminance and chrominance values of the VOP: d[y][x] = p[y][x] + f[y][x]

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

3. Finally, the calculated luminance and chrominance values of the reconstructed VOP are saturated so that

$$d[y][x] = \begin{cases} 2^{bits_per_pixel} - 1; & d[y][x] > 2^{bits_per_pixel} - 1 \\ d[y][x]; & 0 \le d[y][x] \le 2^{bits_per_pixel} - 1 \\ 0; & d[y][x] < 0 \end{cases}$$

7.4 Texture decoding

This subclause describes the process used to decode the texture information of a VOP. The process of video texture decoding is given in Figure 7-3.

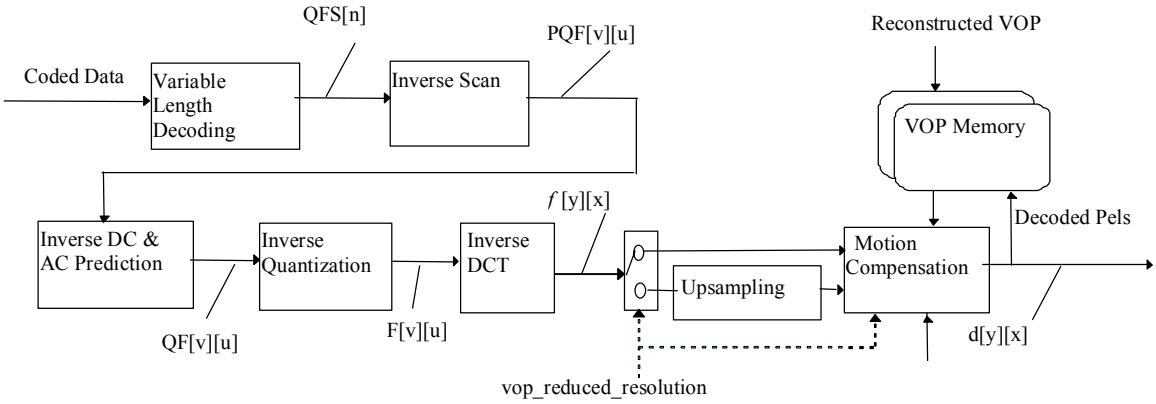


Figure 7-3 — Video Texture Decoding Process

NOTE If flag 'sadct_disable' in the header of VideoObjectLayer() is set to '0', some standard texture decoding techniques from version 1 are modified. In 8x8-blocks with at least one transparent and opaque pel, inverse DCT (subclause 7.4.5) is replaced by inverse shape adaptive DCT (SA-DCT). The number of SA-DCT coefficients in such a block is equal to the number of opaque pels. Only after BAB decoding, it is known to which 8x8-blocks inverse SA-DCT is applied. The usage of inverse SA-DCT in such blocks implies modifications in the processes "Inverse Scan" (subclause 7.3.2), "Inverse AC Prediction" (subclause 7.3.3.3) and "Inverse DCT" (subclause 7.3.5) whereas the processing in the remaining blocks of Figure 7-3 keeps unmodified. The processing of inverse SA-DCT additionally depends on 'mb_type'; i.e.: different types of SA-DCT processing are used in intra- and inter-coded blocks (see subclause 7.3.5).

7.4.1 Variable length decoding

This subclause explains the decoding process. Subclause 7.4.1.1 specifies the process used for the DC coefficients (n=0) in an intra coded block. (n is the index of the coefficient in the appropriate zigzag scan order). Subclause 7.4.1.2 specifies the decoding process for all other coefficients; AC coefficients ($n \neq 0$) and DC coefficients in non-intra coded blocks.

7.4.1.1 DC coefficients decoding in intra blocks

Differential DC coefficients in blocks in intra macroblocks are encoded as variable length code denoting dct_dc_size as defined in Table B.13 and Table B.14 in Annex B, and a fixed length code dct_dc_differential (Table B.15). The dct_dc_size categorizes the DC coefficients according to their "size". For each category additional bits are appended to the dct_dc_size code to uniquely identify which difference in that category actually occurred (Table B.15). This is done by appending a fixed length code, dct_dc_differential, of dct_dc_size bits. The final value of the decoded DC coefficient is the sum of this differential DC value and the predicted value.

When short_video_header is 1, the DC coefficient of an intra block is not coded differentially. It is instead transmitted as a fixed length unsigned integer code of size 8 bits, unless this integer has the value 255. The values 0 and 128 shall not be used – they are reserved. If the integer value is 255, this is interpreted as a signaled value of 128.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4.1.2 Other coefficients

The AC coefficients are obtained by decoding the variable length codes to produce EVENTs. An EVENT is a combination of a last non-zero coefficient indication (LAST; “0”: there are more nonzero coefficients in this block, “1”: this is the last nonzero coefficient in this block), the number of successive zeros preceding the coded coefficient (RUN), and the non-zero value of the coded coefficient (LEVEL).

When short_video_header is 1, the most commonly occurring EVENTS are coded with the variable length codes given in Table B.17 (for all coefficients other than intra DC whether in intra or inter blocks). The last bit “s” denotes the sign of level, “0” for positive and “1” for negative.

When short_video_header is 0, the variable length code table is different for intra blocks and inter blocks. The most commonly occurring EVENTs for the luminance and chrominance components of intra blocks in this case are decoded by referring to the intra columns of Table B.23 when reversible_vlc is set to ‘1’ in I-, P-, or S(GMC)-VOPs, and by referring to Table B.16, otherwise. The most commonly occurring EVENTs for the luminance and chrominance components of inter blocks in this case are decoded by referring to the inter columns of Table B.23 when reversible_vlc is set to ‘1’ in I-, P-, or S(GMC)-VOPs, and by referring to Table B-17, otherwise. The last bit “s” denotes the sign of level, “0” for positive and “1” for negative. The combinations of (LAST, RUN, LEVEL) not represented in these tables are decoded as described in subclause 7.4.1.3.

7.4.1.3 Escape code

The EVENTS that have no variable length code to represent them are coded using an Escape Coding method. The escape codes of DCT coefficients are encoded in five modes. The first three of these modes are used when short_video_header is 0 and in the case that the reversible VLC tables are not used, and the fourth is used when short_video_header is 1. In the case that the reversible VLC tables are used, the fifth escape coding method as in Table B.23 is used. Use of escape sequence of the reversible VLC (Table B.24 and Table B.25) for encoding the combinations listed in Table B.23 is prohibited. Their decoding process is specified below.

Type 1: ESC is followed by “0”, and the code following ESC + ”0” is decoded as a variable length code using the Tcoef VLC codes given in Table B.16 and Table B.17, but the values of LEVEL are modified following decoding to give the restored value $LEVEL^S$, as follows:

$$LEVEL^S = sign(LEVEL^+) \times [\, abs(\, LEVEL^+) + LMAX \,]$$

where $LEVEL^+$ is the value after variable length decoding and LMAX is obtained from Table B.19 and Table B.20 as a function of the decoded values of RUN and LAST.

Type 2: ESC is followed by “10”, and the code following ESC + “10” is decoded as a variable length code using the Tcoef VLC codes given in Table B.16 and Table B.17, but the values of RUN are modified following decoding to give the restored value RUN^S, as follows:

$$RUN^S = RUN^+ + (RMAX + 1)$$

where RUN^+ is the value after variable length decoding. RMAX is obtained from Table B.21 and Table B.22 as a function of the decoded values of LEVEL and LAST.

Type 3: ESC is followed by “11”, and the code following ESC + “11” is decoded as fixed length codes. This type of escape codes are represented by 1-bit LAST, 6-bit RUN and 12-bit LEVEL. A marker bit is inserted before and after the 12-bit-LEVEL in order to avoid the resync_marker emulation. Use of this escape sequence for encoding the combinations listed in Table B.16 and Table B.17 is prohibited. The codes for RUN and LEVEL are given in Table B.18 a and b.

Type 4: The fourth type of escape code is used if and only if short_video_header is 1. In this case, the 15 bits following ESC are decoded as fixed length codes represented by 1-bit LAST, 6-bit RUN and 8-bit LEVEL. The values 0000 0000 and 1000 000 for LEVEL are not used (they are reserved). The codes for RUN and LEVEL are given in Table B.18 a and c.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4.1.4 Intra DC coefficient decoding for the case of switched vlc encoding

At the VOP layer, using quantiser scale value as the threshold, a 3 bit code (intra_dc_vlc_thr) allows switching between 2 VLCs (DC Intra VLC and AC Intra VLC) when decoding DC coefficients of Intra macroblocks, see Table 6.25.

NOTE When the intra AC VLC is turned on, Intra DC coefficients are not handled separately any more, but treated the same as all other coefficients. That means that a zero Intra DC coefficient will not be coded but will only increase the run for the following AC coefficients. The definitions of mcbpc and cbpy in subclause 6.3.6 are changed accordingly.

7.4.2 Inverse scan

This subclause specifies the way in which the one dimensional data, QFS[n] is converted into a two-dimensional array of coefficients denoted by PQF[v][u] where u and v both lie in the range of 0 to 7. Let the data at the output of the variable length decoder be denoted by QFS[n] where n is in the range of 0 to 63. Three scan patterns are defined as shown in Figure 7-4. The scan that shall be used is determined by the following method. For intra blocks, if acpred_flag=0, zigzag scan is selected for all blocks in a macroblock. Otherwise, DC prediction direction is used to select a scan on block basis. For instance, if the DC prediction refers to the horizontally adjacent block, alternate-vertical scan is selected for the current block. Otherwise (for DC prediction referring to vertically adjacent block), alternate-horizontal scan is used for the current block. For all other blocks, the 8x8 blocks of transform coefficients are scanned in the “zigzag” scanning direction.

0	1	2	3	10	11	12	13
4	5	8	9	17	16	15	14
6	7	19	18	26	27	28	29
20	21	24	25	30	31	32	33
22	23	34	35	42	43	44	45
36	37	40	41	46	47	48	49
38	39	50	51	56	57	58	59
52	53	54	55	60	61	62	63

0	4	6	20	22	36	38	52
1	5	7	21	23	37	39	53
2	8	19	24	34	40	50	54
3	9	18	25	35	41	51	55
10	17	26	30	42	46	56	60
11	16	27	31	43	47	57	61
12	15	28	32	44	48	58	62
13	14	29	33	45	49	59	63

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Figure 7-4 — (a) Alternate-Horizontal scan (b) Alternate-Vertical scan (c) Zigzag scan

The inverse scan pattern from Figure 7-4 represent a conversion from the one-dimensional array ‘QFS[n]’ to two-dimensional array ‘PQF[v][u]. This conversion can be expressed as follows where the arrays ‘inv_scan_u[scan_type][n]’ and ‘inv_scan_v[scan_type][n]’ denote the u- and v-components of the selected inverse scan path from Figure 7-4:

```
for (n=0; n<64; n++)
   PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]] = QFS[n];
```

If ‘sadct_disable’ is set to ‘0’, this conversion is modified for those 8x8-blocks in which the number of opaque pels is less than 64:

```
if ((video_object_layer!=”rectangular”) && (sadct_disable==0) &&
  (opaque_pels<64)) {
   coeff_count=0;
   for (n=0; n<64; n++){
      PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]]=0;
      if (coeff_width[inv_scan_v[scan_type][n]] > inv_scan_u[scan_type][n]) {
         PQF[inv_scan_v[scan_type][n]][inv_scan_u[scan_type][n]]=
           QFS[coeff_count];
         coeff_count++;
      }
   }
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTES -

1 After finalization of modified inverse scan the maximal number of non-zero SA-DCT coefficients (see parameter 'coeff_count') equals to 'opaque_pels'. It is absolutely necessary to keep this constraint by setting PQF[v][u] to zero at positions where SA-DCT coefficients are not defined. A non-zero value at these undefined positions of PQF[v][u] would confuse subsequent AC prediction.

2 The permitted values of elements in arrays 'inv_scan_u[scan_type][n]', 'inv_scan_v [scan_type][n]' and 'coeff_width[v]' lie in a range from 0 to 7.

3 Apart from the usage of inverse SA-DCT in subclause 7.3.5 and slight modifications in subclause 7.3.3.3 (inverse AC predicion) the further processing of 'PQF[v][u]' is independent of the setting of flag 'sadct_disable' and keeps unmodified for 'sadct_disable=0'.

The above definition of the modified inverse scan requires an auxiliary array 'coeff_ width[v]' which has to be derived from the decoded BAB. It indicates the number of SA-DCT coefficients which are available in each particular row of PQF[v][u]. See Annex A (subclause A.3.1) for details.

7.4.3 DC and AC prediction for intra macroblocks

This subclause specifies the prediction process for decoding of coefficients. This prediction process is only carried out for intra-macroblocks (I-MBs) and when short_video_header is "0". When short_video_header is "1" or the macroblock is not an I-MB, this prediction process is not performed.

7.4.3.1 DC and AC Prediction Direction

This adaptive selection of the DC and AC prediction direction is based on comparison of the horizontal and vertical DC differences around the block to be decoded. Figure 7-5 shows the three blocks surrounding the block to be decoded. Block 'X', 'A', 'B' and 'C' respectively refer to the current block, the left block, the above-left block, and the block immediately above, as shown.

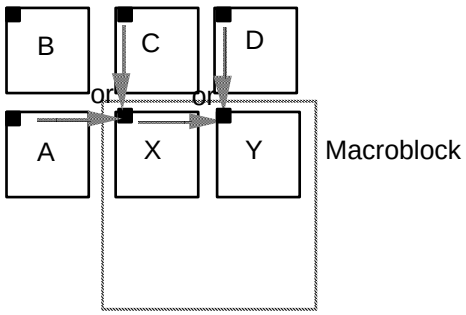


Figure 7-5 — Previous neighboring blocks used in DC prediction

The inverse quantised DC values of the previously decoded blocks, F[0][0], are used to determine the direction of the DC and AC prediction as follows.

if ($|F_A[0][0] - F_B[0][0]| < |F_B[0][0] - F_C[0][0]|$)

predict from block C

else

predict from block A

If any of the blocks A, B or C are outside of the VOP boundary, or the video packet boundary, or they do not belong to an intra coded macroblock, their F[0][0] values are assumed to take a value of $2^{(bits_per_pixel+2)}$ and are used to compute the prediction values.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4.3.2 Adaptive DC Coefficient Prediction

The adaptive DC prediction method involves selection of either the F[0][0] value of immediately previous block or that of the block immediately above it (in the previous row of blocks) depending on the prediction direction determined above.

if (predict from block C)

$QF_X[0][0] = PQF_X[0][0] + F_C[0][0]$ // dc_scaler

else

$QF_X[0][0] = PQF_X[0][0] + F_A[0][0]$ // dc_scaler

dc_scalar is defined in Table 7-1. This process is independently repeated for every block of a macroblock using the appropriate immediately horizontally adjacent block 'A' and immediately vertically adjacent block 'C'.

DC predictions are performed similarly for the luminance and each of the two chrominance components.

7.4.3.3 Adaptive AC coefficient prediction

This process is used when ac_pred_flag = '1', which indicates that AC prediction is performed when decoding the coefficients.

Either coefficients from the first row or the first column of a previous coded block are used to predict the co-sited coefficients of the current block. On a block basis, the direction (from among horizontal and vertical directions) for DC coefficient prediction is also used to select the direction for AC coefficients prediction. The AC coefficients prediction is illustrated in Figure 7-6.

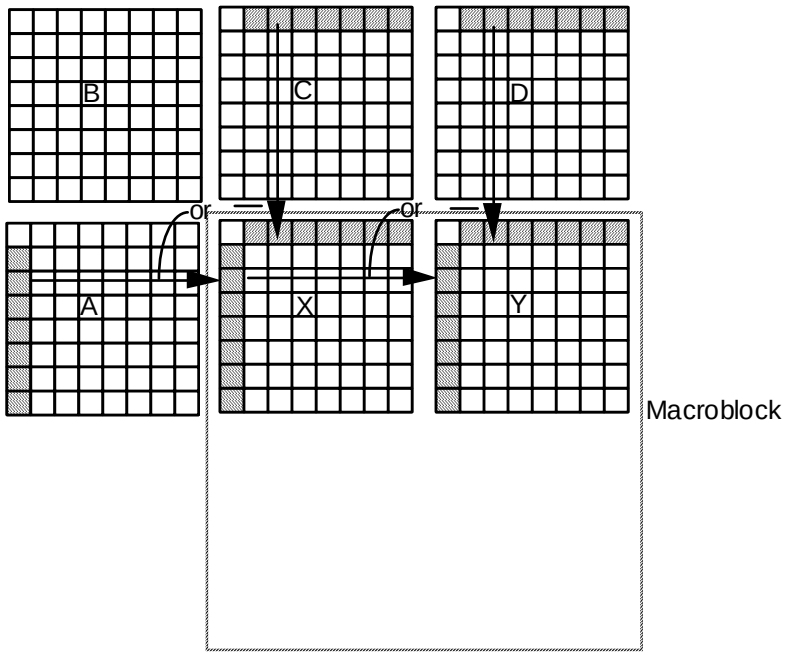


Figure 7-6 — Previous neighboring blocks and coefficients used in AC prediction

To compensate for differences in the quantisation of previous horizontally adjacent or vertically adjacent blocks used in AC prediction of the current block, scaling of prediction coefficients becomes necessary. Thus the prediction is modified so that the predictor is scaled by the ratio of the current quantiser scale and the quantiser scale of the predictor block. The definition is given in the equations below.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If block 'A' was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first column of the quantised AC coefficients as follows.

$$QF_X[v][0] = PQF_X[v][0] + (QF_A[v][0] * Qp_A) \,//\, Qp_X \qquad v = 1 \text{ to } 7$$

If block 'C' was selected as the predictor for the block for which coefficient prediction is to be performed, calculate the first row of the quantised AC coefficients as follows.

$$QF_X[0][u] = PQF_X[0][u] + (QF_C[0][u] * Qp_C) \,//\, Qp_X \qquad u = 1 \text{ to } 7$$

If the prediction block (block 'A' or block 'C') is outside of the boundary of the VOP or video packet, then all the prediction coefficients of that block are assumed to be zero. Qp_A, Qp_C and Qp_X are the quantiser scale values of macroblocks A, C and X respectively.

When AC coefficient prediction is used in conjunction with 'sadct_disable=0', the results of AC prediction has to be post-processed by shape parameters. This shape-adaptive post-processing takes care that those elements of 'QF[v][u]', which are not defined in SA-DCT and which should be zero therefore, are again set to zero after AC prediction. An auxiliary array 'coeff_ width[v]' which has to be derived from the decoded BAB (see subclause A.3.1 in Annex A for details) is used for this purpose: in column 'u=0' ('A'-predictor) or in row 'v=0' ('C'-predictor) all elements of 'QF[v][u]' with 'coeff_width[v] = 0' or 'u >= coeff_width[0]' are set to zero after AC prediction

7.4.3.4 Saturation of QF[v][u]

The quantised coefficients resulting from the DC and AC Prediction are saturated to lie in the range [−2048, 2047]. Thus:

$$QF[v][u] = \begin{cases} 2047 & QF[v][u] > 2047 \\ QF[v][u] & -2048 \le QF[v][u] \le 2047 \\ -2048 & QF[v][u] < -2048 \end{cases}$$

7.4.4 Inverse quantisation

The two-dimensional array of coefficients, *QF*[*v*][*u*], is inverse quantised to produce the reconstructed DCT coefficients. This process is a multiplication by the quantisation step size. The quantisation step size is modified by two mechanisms; a quantisation matrix is used to modify the step size within a block and a quantiser scale is used in order that the step size can be modified at the cost of only a few bits (as compared to encoding an entire new quantisation matrix).

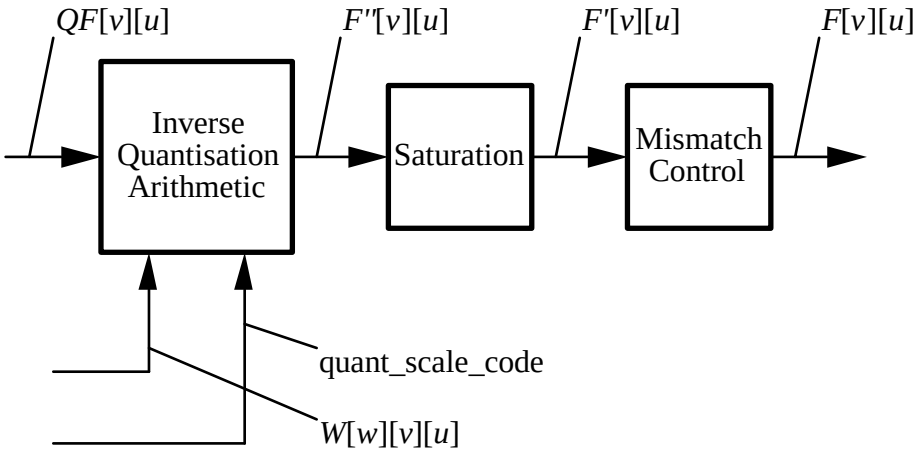


Figure 7-7 — Inverse quantisation process

Figure 7-7 illustrates the overall inverse quantisation process. After the appropriate inverse quantisation arithmetic the resulting coefficients, *F''*[*v*][*u*], are saturated to yield *F'*[*v*][*u*] and then a mismatch control operation is performed to give the final reconstructed DCT coefficients, *F*[*v*][*u*].

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

NOTE Attention is drawn to the fact that the method of achieving mismatch control in this part of ISO/IEC 14496 is identical to that employed by ISO/IEC 13818-2:2000.

7.4.4.1 First inverse quantisation method

This subclause specifies the first of the two inverse quantisation methods. The method described here is used when quant_type equals 1.

7.4.4.1.1 Intra DC coefficient

The DC coefficients of intra coded blocks shall be inverse quantised in a different manner to all other coefficients.

In intra blocks *F'*[0][0] shall be obtained by multiplying *QF*[0][0] by a constant multiplier.

The reconstructed DC values are computed as follows.

F'[0][0] = *dc_scaler* QF*[0][0]

When short_video_header is 1, dc_scaler is 8, otherwise dc_scaler is defined in Table 7-1.

7.4.4.1.2 Other coefficients

All coefficients other than the DC coefficient of an intra block shall be inverse quantised as specified in this subclause. Two quantisation matrices are used. One shall be used for intra macroblocks and the other for non-intra macroblocks. Each matrix has a default set of values which may be overwritten by down-loading a user defined matrix.

Let the quantisation matrices be denoted by *W*[*w*][*v*][*u*] where *w* takes the values 0 to 1 indicating which of the matrices is being used. *W*[*0*][*v*][*u*] is for intra macroblocks, and *W*[*1*][*v*][*u*] is for non-intra macroblocks. The value of *quantiser_scale* is determined by vop_quant, dquant, dbquant, and quant_scale for luminance and chrominance, and additionally by vop_quant_alpha for grayscale alpha. For example, the value of *quantiser scale* for luminance and chrominance shall be an integer from 1 to 31 when not_8_bit == '0'. The following equation specifies the arithmetic to reconstruct *F''*[*v*][*u*] from *QF*[*v*][*u*] (for all coefficients except intra DC coefficients).

$$F''[v][u] = \begin{cases} 0, & \text{if } QF[v][u] = 0 \\ ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser_scale) \;/\; 16, & \text{if } QF[v][u] \neq 0 \end{cases}$$

where :

$$k = \begin{cases} 0 & \text{intra blocks} \\ Sign(QF[v][u]) & \text{non - intra blocks} \end{cases}$$

NOTE The above equation uses the "/" operator as defined in subclause 4.1.

7.4.4.2 Second inverse quantisation method

This subclause specifies the second of the two inverse quantisation methods. The method described here is used for all the coefficients other than the DC coefficient of an intra block when quant_type==0. In the second inverse quantisation method, the DC coefficient of an intra block is quantised using the same method as in the first inverse quantisation method (see subclause 7.4.4.1.1). The quantiser scale *quantiser_scale* may take integer values from 1 to $2^{quant_precision}-1$.

7.4.4.2.1 Inverse quantisation

$$|F''[v][u]| = \begin{cases} 0, & \text{if } QF[v][u] = 0, \\ (2 \times |QF[v][u]| + 1) \times quantiser_scale, & \text{if } QF[v][u] \neq 0,\ quantiser_scale \text{ is odd}, \\ (2 \times |QF[v][u]| + 1) \times quantiser_scale - 1, & \text{if } QF[v][u] \neq 0,\ quantiser_scale \text{ is even}. \end{cases}$$

The sign of *QF*[*v*][*u*] is then incorporated to obtain *F''*[*v*][*u*]: *F''*[*v*][*u*]= Sign(*QF*[*v*][*u*])×|*F''*[*v*][*u*]|.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4.4.3 Nonlinear inverse DC quantisation

NOTE This subclause is valid for both quantisation methods.

Within an Intra macroblock for which short_video_header is 0, luminance blocks are called type 1 blocks, chroma blocks are classified as type 2. When short_video_header is 1, the inverse quantisation of DC intra coefficients is equivalent to using a fixed value of dc_scaler = 8, as described above in subclause 7.4.1.1.

- DC coefficients of Type 1 blocks are quantised by Nonlinear Scaler for Type 1
- DC coefficients of Type 2 blocks are quantised by Nonlinear Scaler for Type 2

Table 7-1 specifies the nonlinear dc_scaler expressed in terms of piece-wise linear characteristics.

Table 7-1 — Non linear scaler for DC coefficients of DCT blocks, expressed in terms of relation with quantiser_scale

Component:Type	dc_scaler for quantiser_scale range			
	1 through 4	5 through 8	9 through 24	>= 25
Luminance: Type1	8	2x quantiser_scale	quantiser_scale +8	2 x quantiser_scale -16
Chrominance: Type2	8	(quantiser_scale +13)/2		quantiser_scale -6

7.4.4.4 Saturation

The coefficients resulting from the inverse quantisation arithmetic are saturated to lie in the range $[-2^{bits_per_pixel + 3}, 2^{bits_per_pixel + 3} - 1]$. Thus:

$$F'[v][u] = \begin{cases} 2^{bits_per_pixel+3} - 1 & F''[v][u] > 2^{bits_per_pixel+3} - 1 \\ F''[v][u] & -2^{bits_per_pixel+3} \le F''[v][u] \le 2^{bits_per_pixel+3} - 1 \\ -2^{bits_per_pixel+3} & F''[v][u] < -2^{bits_per_pixel+3} \end{cases}$$

7.4.4.5 Mismatch control

This mismatch control is only applicable to the first inverse quantisation method. Mismatch control shall be performed by any process equivalent to the following. Firstly all of the reconstructed, saturated coefficients, *F'[v][u]* in the block shall be summed. This value is then tested to determine whether it is odd or even. If the sum is even then a correction shall be made to just one coefficient; *F*[7][7]. Thus:

$$sum = \sum_{v=0}^{v<8} \sum_{u=0}^{u<8} F'[v][u]$$

$$F[v][u] = F'[v][u] \text{ for all } u, v \text{ except } u = v = 7$$

$$F[7][7] = \begin{cases} F'[7][7] & \text{if } sum \text{ is odd} \\ \left\{ \begin{array}{ll} F'[7][7] - 1 & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1 & \text{if } F'[7][7] \text{ is even} \end{array} \right\} & \text{if } sum \text{ is even} \end{cases}$$

NOTE 1 It may be useful to note that the above correction for F[7][7] may be implemented by toggling the least significant bit of the twos complement representation of the coefficient. Also since only the “oddness” or “evenness” of the sum is of interest an exclusive OR (of just the least significant bit) may be used to calculate “sum”.

NOTE 2 Warning. Small non-zero inputs to the IDCT may result in zero output for compliant IDCTs. If this occurs in an encoder, mismatch may occur in some pictures in a decoder that uses a different compliant IDCT. An encoder should avoid this problem and may do so by checking the output of its own IDCT. It should ensure that it never inserts any non-zero coefficients into the bitstream when the block in question reconstructs to zero through its own IDCT function. If this action is not taken by the encoder, situations can arise where large and very visible mismatches between the state of the encoder and decoder occur.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4.4.6 Summary of quantiser process for method 1

In summary, the method 1 inverse quantisation process is any process numerically equivalent to:

```
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if (QF[v][u] == 0)
            F''[v][u] = 0;
        else if ( (u==0) && (v==0) && (macroblock_intra) ) {
            F''[v][u] = dc_scaler * QF[v][u];
        } else {
            if ( macroblock_intra ) {
                F''[v][u] = ( QF[v][u] * W[0][v][u] * quantiser_scale * 2 ) / 16;
            } else {
                F''[v][u] = ( ( ( QF[v][u] *  2 ) + Sign(QF[v][u]) ) * W[1][v][u]
                                                        * quantiser_scale ) / 16;
            }
        }
    }
}

sum = 0;
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if ( F''[v][u] > 2^(bits_per_pixel + 3) – 1 ) {
            F'[v][u] = 2^(bits_per_pixel + 3) – 1;
        } else {
            if ( F''[v][u] < -2^(bits_per_pixel + 3) ) {
                F'[v][u] = -2^(bits_per_pixel + 3);
            } else {
                F'[v][u] = F''[v][u];
            }
        }
    sum = sum + F'[v][u];
    F[v][u] = F'[v][u];
    }
}

if ((sum & 1) == 0) {
    if ((F[7][7] & 1) != 0) {
        F[7][7] = F'[7][7] - 1;
    } else {
        F[7][7] = F'[7][7] + 1;
    }
}
```

7.4.5 Inverse DCT

Once the DCT coefficients, F[u][v] are reconstructed, the inverse DCT transform defined in Annex A shall be applied to obtain the inverse transformed values, $f[y][x]$. These values shall be saturated so that: $-2^{bits_per_pixel} \leq f[y][x] \leq 2^{bits_per_pixel} - 1$, for all x, y.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If flag 'sadct_disable' in the header of VideoObjectLayer() is set to '0', inverse 8x8-DCT – as specified in version 1 - is only applied to 8x8-blocks with 64 opaque pels. In 8x8-blocks with a least one transparent pel (i.e. number of opaque pels is less than 64), the inverse DCT is replaced by inverse shape adaptive DCT (SA-DCT). Similar to 8x8-DCT, SA-DCT reconverts F[v][u] to f[x][y]. The internal processing of inverse SA-DCT is controlled by special shape parameters which have to be computed from decoded BAB. An extended version of SA-DCT, called ΔDC-SA-DCT, is used in intra-coded 8x8-blocks with 'opaque_pels<64'. The decision on the transform to be used in the current 8x8-block depends on the settings of 'video_object_layer_shape', 'sadct_disable', 'vop_coding_type' and 'derived_mb_type' as well as parameter 'opaque_pels'. The decision rule is specified in Table 7-2. Details on internal SA-DCT and ΔDC-SA-DCT processing are given in subclauses 10.3 and 10.4 of Annex A.

Table 7-2 — Decision rule for the using 8x8-DCT, SA-DCT and ΔDC-SA-DCT

decision rules	DCT tool
(video_object_layer_shape=="rectangular") || (sadct_disable==1) || (opaque_pels== 64)	8x8-DCT
(video_object_layer_shape !="rectangular") && (sadct_disable==0) && (opaque_pels <64) && (vop_coding_type !="B") && ((derived_mb_type==3) || (derived_mb_type==4))	ΔDC-SA-DCT
(video_object_layer_shape !="rectangular") && (sadct_disable==0) && (opaque_pels < 64) && (((vop_coding_type ="P") && (derived_mb_type!=3) && (derived_mb_type != 4)) || (vop_coding_type ="B"))	SA-DCT

7.4.6 Upsampling of the Inverse DCT output for Reduced Resolution VOP

For the VOP with vop_reduced_resolution flag equal to "1", the upsampling process described in this subclause shall be applied to the Inverse DCT output $f[y][x]$. When the flag is set to "0" or the flag is not present, this upsampling process shall be skipped.

The 16×16 reconstructed prediction error block is obtained by upsampling the 8×8 reconstructed prediction error block $f[y][x]$. In order to realize a simple implementation, filtering is closed within a block which enables to perform an individual upsampling on block basis. Figure 7-8 shows the positioning of samples. The upsampling procedure for the luminance and chrominance pixels which are inside the 16×16 reconstructed prediction error blocks is defined in subclause 7.4.6.1. For the creation of the luminance and chrominance pixels which are at the boundary of 16×16 reconstructed prediction error block, the procedure is defined in subclause 7.4.6.2. Chrominance blocks as well as luminance blocks are up-sampled. The upsampled 16 x 16 prediction error blocks are used for the motion compensation decoding process for Reduced Resolution VOP described in the subclause 7.6.10.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

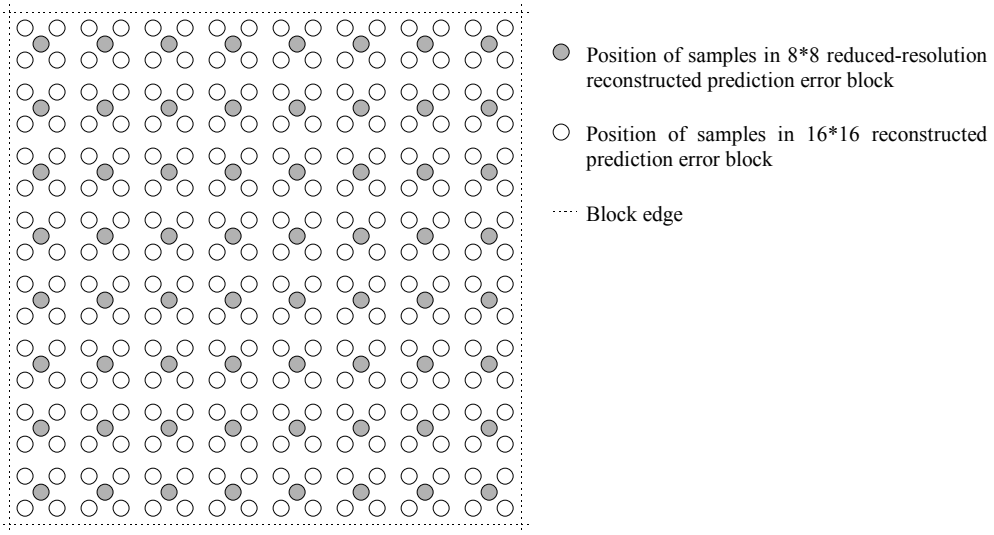


Figure 7-8 — Positioning of samples in 8×8 reduced-resolution reconstructed prediction error block and 16×16 reconstructed prediction error block

7.4.6.1 Upsampling procedure for the pixels inside a 16x16 reconstructed prediction error block

The creation of reconstructed prediction error for pixels inside block is described in Figure 7-9.

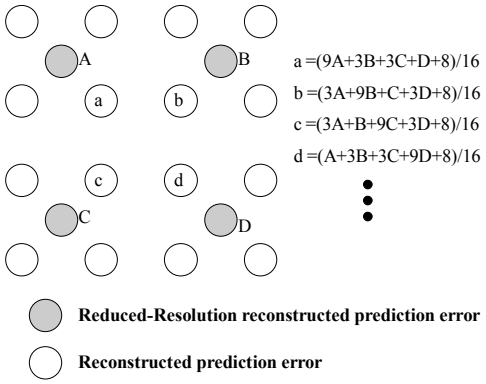


Figure 7-9 — Creation of reconstructed prediction error for pixels inside block

7.4.6.2 Upsampling procedure for the pixels at the boundary of 16x16 reconstructed prediction error block

The creation of reconstructed prediction error for pixels of a 16x16 block is shown in Figure 7-10.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

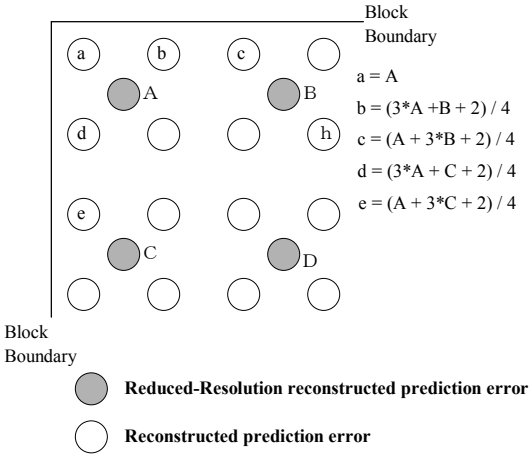


Figure 7-10 — Creation of reconstructed prediction error for pixels at the block boundary

7.5 Shape decoding

Binary shape decoding is based on a block-based representation. The primary coding methods are block-based context-based binary arithmetic decoding and block-based motion compensation. The primary data structure used is denoted as the binary alpha block (bab). The bab is a square block of binary valued pixels representing the opacity/transparency for the pixels in a specified block-shaped spatial region of size 16x16 pels. In fact, each bab is co-located with each texture macroblock.

7.5.1 Higher syntactic structures

7.5.1.1 VOL decoding

If video_object_layer_shape is equal to '00' then no binary shape decoding is required. Otherwise, binary shape decoding is carried out.

7.5.1.2 VOP decoding

If video_object_layer_shape is not equal to '00' then, for each subsequent VOP, the dimensions of the bounding rectangle of the reconstructed VOP are obtained from:

- vop_width
- vop_height

If these decoded dimensions are not multiples of 16, then the values of vop_width and vop_height are rounded up to the nearest integer, which is a multiple of 16.

Additionally, in order to facilitate motion compensation, the horizontal and spatial position of the VOP are obtained from:

- vop_horizontal_mc_spatial_ref
- vop_vertical_mc_spatial_ref

These spatial references may be different for each VOP but the same coordinate system must be used for all VOPs within a vol. Additionally, the decoded spatial references must have an even value.

- vop_shape_coding_type

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This flag enables the use of intra shape codes in P, B, or S(GMC)-VOPs. Finally, in the VOP class, it is necessary to decode

- change_conv_ratio_disable

This specifies whether conv_ratio is encoded at the macroblock layer.

Once the above elements have been decoded, the binary shape decoder may be applied to decode the shape of each macroblock within the bounding rectangle.

7.5.2 Macroblock decoding

The shape information for each macroblock residing within the bounding rectangle of the VOP is decoded into the form of a 16x16 bab.

7.5.2.1 Mode decoding

Each bab belongs to one of seven types listed in Table 7-3. The type information is given by the bab_type field which influences decoding of further shape information. For I-VOPs only three out of the seven modes are allowed as shown in Table 7-3.

Table 7-3 — List of bab types

bab_type	Semantic	Used in
0	MVDs==0 && No Update	P- ,B-, and S(GMC)-VOPs
1	MVDs!=0 && No Update	P- , B-, and S(GMC)-VOPs
2	Transparent	All VOP types
3	Opaque	All VOP types
4	IntraCAE	All VOP types
5	MVDs==0 && interCAE	P- , B-, and S(GMC)-VOPs
6	MVDs!=0 && interCAE	P- , B-, and S(GMC)-VOPs

7.5.2.1.1 I-VOPs

Suppose that `f(x,y)` is the bab_type of the bab located at `(x,y)`, where x is the BAB column number and y is the BAB row number. The code word for the bab_type at the position `(i,j)` is determined as follows. A context C is computed from previously decoded bab_type's.

C = 27*(f(i-1,j-1)-2) + 9*(f(i,j-1)-2) + 3*(f(i+1,j-1)-2) + (f(i-1,j)-2)

If `f(x,y)` references a bab outside the current VOP, bab_type is assumed to be transparent for that bab (i.e. `f(x,y)=2`). The bab_type of babs outside the current video packet is also assumed to be transparent. The VLC used to decode bab_type for the current bab is switched according to the value of the context C. This context-switched VLC table is given in Table B-27.

7.5.2.1.2 P- and B-, and S(GMC)-VOPs

The decoding of the current bab_type is dependent on the bab_type of the co-located bab in the reference VOP. The reference VOP is either a forward reference VOP or a backward reference VOP. The forward reference VOP is defined as the most recent non-empty (i.e. vop_coded != 0) I- or P-, or S(GMC)-VOP in the past, while the backward VOP is defined as the most recently decoded I- or P-, or S(GMC)-VOP in the future. If the current VOP is a P-, or S(GMC)-VOP, the forward reference VOP is selected as the reference VOP. If the current VOP is a B-VOP the following decision rules are applied:

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1. If the backwards reference VOP is empty, the non-empty one (forward) is selected as the reference VOP for the current B-VOP.

2. If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.

If the sizes of the current and reference VOPs are different, some babs in the current VOP may not have a co-located equivalent in the reference VOP. Therefore the bab_type matrix of the reference VOP is manipulated to match the size of the current VOP. Two rules are defined for that purpose, namely a cut rule and a copy rule:

- ***cut rule***. If the number of lines (respectively columns) is smaller in the current VOP than in the reference VOP, the bottom lines (respectively rightmost columns) are eliminated from the reference VOP such that both VOP sizes match.
- ***copy rule***. If the number of lines (respectively columns) is larger in the current VOP than in the reference VOP, the bottom line (respectively rightmost column) is replicated as many times as needed in the reference VOP such that both VOP sizes match.

An example is shown in Figure 7-11 where both rules are applied.

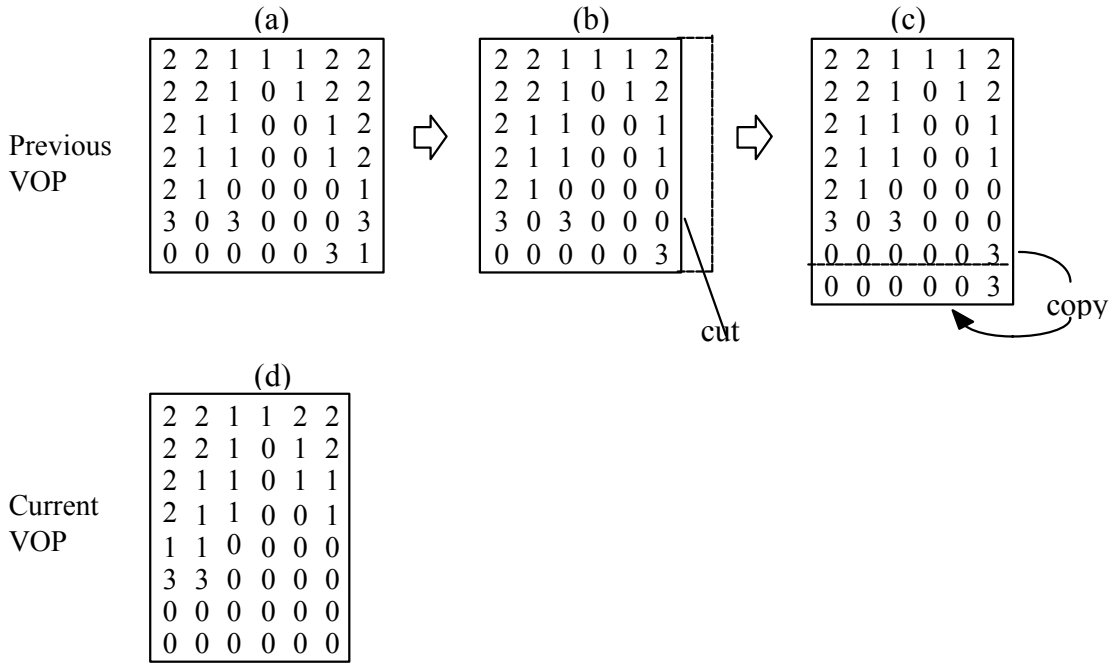


Figure 7-11 — Example of size fitting between current VOP and reference VOP. The numbers represent the type of each bab

The VLC to decode the current bab_type is switched according to the value of bab_type of the co-located bab in the reference VOP. This context-switched VLC tables for P, B, and S(GMC)-VOPs are given in Table B.28. If the type of the bab is transparent, then the current bab is filled with zero (transparent) values. A similar procedure is carried out if the type is opaque, where the reconstructed bab is filled with values of 255 (opaque). For both transparent and opaque types, no further decoding of shape-related data is required for the current bab. Otherwise further decoding steps are necessary, as listed in Table 7-4. Decoding for motion compensation is described in subclause 7.5.2.2, and cae decoding in subclause 7.5.2.5.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-4 — Decoder components applied for each type of bab

bab_type	Motion compensation	CAE decoding
0	yes	no
1	yes	no
2	no	no
3	no	no
4	no	yes
5	yes	yes
6	yes	yes

7.5.2.2 Binary alpha block motion compensation

Motion Vector of shape (MVs) is used for motion compensation (MC) of shape. The value of MVs is reconstructed as described in subclause 7.5.2.3. Unrestricted motion compensation is applied, however, Overlapped MC, half sample MC and 8x8 MC are not carried out. Integer pixel unrestricted motion compensation is carried out on a 16x16 block basis according to subclause 7.5.2.4.

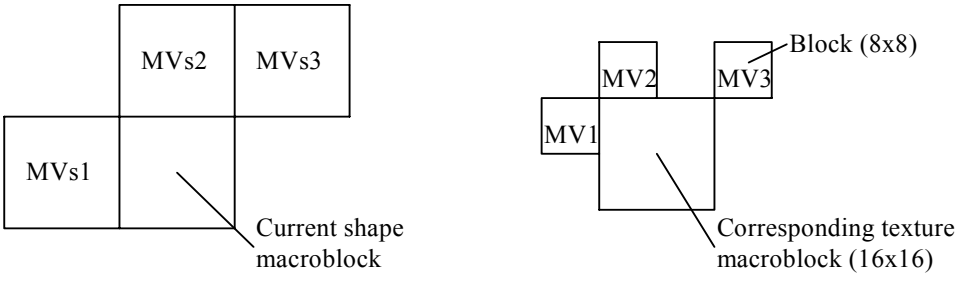
If bab_type is MVDs==0 && No Update or MVDs!=0 && No Update then the motion compensated bab is taken to be the decoded bab, and no further decoding of the bab is necessary. Otherwise, cae decoding is required.

7.5.2.3 Motion vector decoding

If bab_type indicates that MVDs!=0, then mvds_x and mvds_y are VLC decoded. For decoding mvds_x, the VLC given in Table B.29 is used. The same table is used for decoding mvds_y, unless the decoded value of mvds_x is zero. If mvds_x == 0, the VLC given in Table B.30 is used for decoding mvds_y. If bab_type indicates that MVDs==0, then both mvds_x and mvds_y are set to zero.

The integer valued shape motion vector MVs=(mvs_x,mvs_y) is determined as the sum of a predicted motion vector MVPs and MVDs = (mvds_x,mvds_y), where MVPs is determined as follows.

MVPs is determined by analysing certain candidate motion vectors of shape (MVs) and motion vectors of selected texture blocks (MV) around the MB corresponding to the current bab. They are located and denoted as shown in Figure 7-12 where MV1, MV2 and MV3 are rounded up to integer values towards 0. If the selected texture block is a field predicted macroblock, then MV1, MV2 or MV3 are generated by averaging the two field motion vectors and rounding toward zero. Regarding the texture MV's, the convention is that a MB possessing only 1 MV is considered the same as a MB possessing 4 MV's, where the 4 MV's are equal. By traversing MVs1, MVs2, MVs3, MV1, MV2 and MV3 in this order, MVPs is determined by taking the first encountered MV that is defined. If no candidate motion vectors is defined, MVPs = (0,0).



(1) MV for shape (2) MV for texture

Figure 7-12 — Candidates for MVPs

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the case that video_object_layer_shape is “binary_only” or vop_coding_type indicates B-VOP, MVPs is determined by considering the motion vectors of shape (MVs1, MVs2 and MVs3) only. The following subclauses explain the definition of MVs1, MVs2, MVs3, MV1, MV2 and MV3 in more detail.

Defining candidate predictors from texture motion vectors:

One shape motion vector predictor MV_i (i =1,2,3) is defined for each block located around the current bab according to Figure 7-12 (2). The definition only depends on the transparency of the reference MB. MVi is set to the corresponding block vector as long as it is in a non-transparent reference MB, otherwise, it is not defined. Note that if a reference MB is outside the current VOP or video packet, it is treated as a transparent MB.

Defining candidate predictors from shape motion vectors:

The candidate motion vector predictors MVs_i are defined by the shape motion vectors of neighbouring bab located according to Figure 7-12 (1). The MVs_i are defined according to Table 7-5.

Table 7-5 — Definition of candidate shape motion vector predictors MVs1, MVs2, and MVs3 from shape motion vectors for P, B, and S(GMC)-VOPs. Note that interlaced modes are not included

Shape mode of reference MB	**MVs_i for each reference shape block-i (a shape block is 16x16)**
MVDs == 0 or MVDs !=0 bab_type 0, 1, 5,6	The retrieved shape motion vector of the said reference MB is defined as MVs_i. Note that MVs_i is defined, and hence valid, even if the reconstructed shape block is transparent.
all_0, bab_type 2	MVs_i is undefined
all=255, bab_type 3	MVs_i is undefined
Intra, bab_type 4	MVs_i is undefined

If the reference MB is outside of the current video packet, MV_i and MVs_i are undefined.

7.5.2.4 Motion compensation

For inter mode babs (bab_type = 0,1,5 or 6), motion compensation is carried out by simple MV displacement according to the MVs.

Specifically, when bab_type is equal to 0 or 1 i.e. for the no-update modes, a displaced block of 16x16 pixels is copied from the binary alpha map of the previously decoded I or P-, or S(GMC)- VOP for which vop_coded is not equal to ‘0’. When the bab_type is equal to 5 or 6 i.e. when interCAE decoding is required, then the pixels immediately bordering the displaced block (to the left, right, top and bottom) are also copied from the most recent valid reference VOP’s (as defined in subclause 6.3.5) binary alpha map into a temporary shape block of 18x18 pixels size (see Figure 7-15). If the displaced position is outside the bounding rectangle, then these pixels are assumed to be “transparent”.

If the current VOP is a B-VOP the following decision rules are applied:

- If the backwards reference VOP is empty, the non-empty one (forward) is selected as the reference VOP for the current B-VOP.

- If both reference VOPs are non-empty, the forward reference VOP is selected if its temporal distance to the current B-VOP is not larger than that of the backward reference VOP, otherwise, the backward one is chosen.

In the special cases when closed_gov == 1 and the forward reference VOP belongs to the previous GOV, the current B-VOP takes the backward VOP as reference.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.2.5 Context based arithmetic decoding

Before decoding the binary_arithmetic_code field, border formation (see subclause 7.5.2.5.2) needs to be carried out. Then, if the scan_type field is equal to 0, the bordered to-be decoded bab and the eventual bordered motion compensated bab need to be transposed (as for matrix transposition). If change_conv_rate_disable is equal to 0, then conv_ratio is decoded to determine the size of the sub-sampled BAB, which is 16/conv_ratio by 16/conv_ratio pixels large. If change_conv_rate_disable is equal to 1, then the decoder assumes that the bab is not subsampled and thus the size is 16x16 pixels. Binary_arithmetic_code is then decoded by a context-based arithmetic decoder as follows. The arithmetic decoder is firstly initialised (see subclause 7.5.3.3). The pixels of the sub-sampled bab are decoded in raster order. At each pixel,

1. A context number is computed based on a template, as described in subclause 7.5.2.5.1.
2. The context number is used to access the probability table (Table B.32).
3. Using the accessed probability value, the next bits of binary_arithmetic_code are decoded by the arithmetic decoder to give the decoded pixel value.

When all pixels in sub-sampled BAB have been decoded, the arithmetic decoder is terminated (see subclause 7.5.3.6).

If the scan_type field is equal to 0, the decoded bab is transposed. Then up-sampling is carried out if conv_ratio is different from 1, as described in subclause 7.5.2.5.3. Then the decoded bab is copied into the decoded shape map.

7.5.2.5.1 Context computation

For INTRA coded BABs, a 10 bit context $C = \sum_k c_k \cdot 2^k$ is built for each pixel as illustrated in Figure 7-13 (a), where c_k=0 for transparent pixels and c_k=1 for opaque pixels.

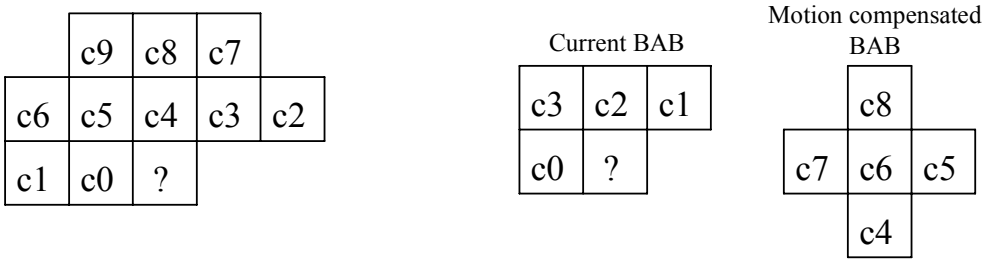


Figure 7-13 — (a) The INTRA template (b) The INTER template where c6 is aligned with the pixel to be decoded. The pixel to be decoded is marked with '?'

For INTER coded BABs, temporal redundancy is exploited by using pixels from the bordered motion compensated BAB (depicted in Figure 7-15) to make up part of the context. Specifically, a 9 bit context $C = \sum_k c_k \cdot 2^k$ is built as illustrated in Figure 7-13 (b).

There are some special cases to note.

- When building contexts, any pixels outside the bounding rectangle of the current VOP to the left and above are assumed to be zero (transparent).
- When building contexts, any pixels outside the space of the current video packet to the left and above are assumed to be zero (transparent).
- The template may cover pixels from BABs which are unknown at decoding time. Unknown pixels are defined as area U in Figure 7-14.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- The values of these unknown pixels are defined by the following procedure:
    - When constructing the INTRA context, the following steps are taken in the sequence

        1. if (c7 is unknown) c7=c8,

        2. if (c3 is unknown) c3=c4,

        3. if (c2 is unknown) c2=c3.

    - When constructing the INTER context, the following conditional assignment is performed.

    if (c1 is unknown) c1=c2

7.5.2.5.2 Border formation

When decoding a BAB, pixels from neighbouring BABs shall be used to make up the context. For both the INTRA and INTER cases, a 2 pixel wide border about the current BAB is used where pixels values are known, as depicted in Figure 7-14.

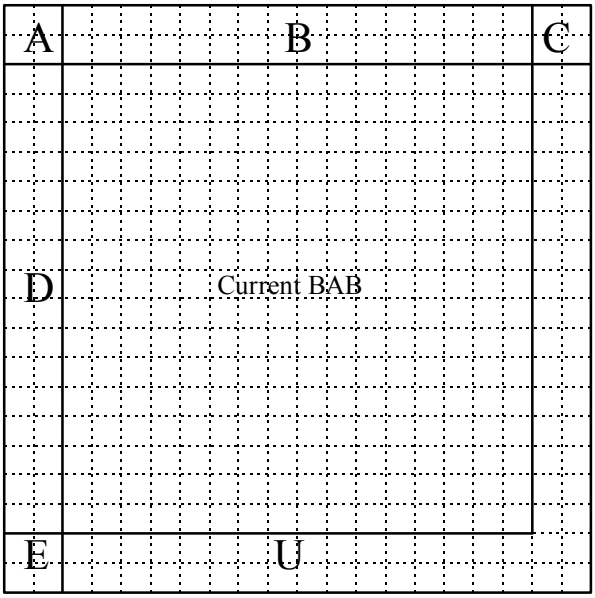


Figure 7-14 — Bordered BAB. A: TOP_LEFT_BORDER. B: TOP_BORDER. C: TOP_RIGHT_BORDER. D: LEFT_BORDER. E: BOTTOM_LEFT_BORDER. U: pixels which are unknown when decoding the current BAB

If the value of conv_ratio is not equal to 1, a sub-sampling procedure is further applied to the BAB borders for both the current BAB and the motion compensated BAB.

The border of the current BAB is partitioned into 5 regions:

- TOP_LEFT_BORDER, which contains pixels from the BAB located to the upper-left of the current BAB and which consists of 2 lines of 2 pixels
- TOP_BORDER, which contains pixels from the BAB located above the current BAB and which consists of 2 lines of 16 pixels
- TOP_RIGHT_BORDER, which contains pixels from the BAB located to the upper-right of the current BAB and which consists of 2 lines of 2 pixels
- LEFT_BORDER, which contains pixels from the BAB located to the left of the current BAB and which consists of 2 columns of 16 pixels

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- BOTTOM_LEFT_BORDER, which contains pixels from the BAB located to the bottom-left of the current BAB and which consists of 2 lines of 2 pixels

The TOP_LEFT_BORDER and TOP_RIGHT_BORDER are not sub-sampled, and kept as they are. The TOP_BORDER and LEFT_BORDER are sub-sampled such as to obtain 2 lines of 16/conv_ratio pixels and 2 columns of 16/conv_ratio pixels, respectively.

The sub-sampling procedure is performed on a line-basis for TOP_BORDER, and a column-basis for LEFT_BORDER. For each line (respectively column), the following algorithm is applied: the line (respectively column) is split into groups of conv_ratio pixels. For each group of pixels, one pixel is associated in the sub-sampled border. The value of the pixel in the sub-sampled border is OPAQUE if half or more pixels are OPAQUE in the corresponding group. Otherwise the pixel is TRANSPARENT.

The 2x2 BOTTOM_LEFT_BORDER is filled by replicating downwards the 2 bottom border samples of the LEFT_BORDER after the down-sampling (if any).

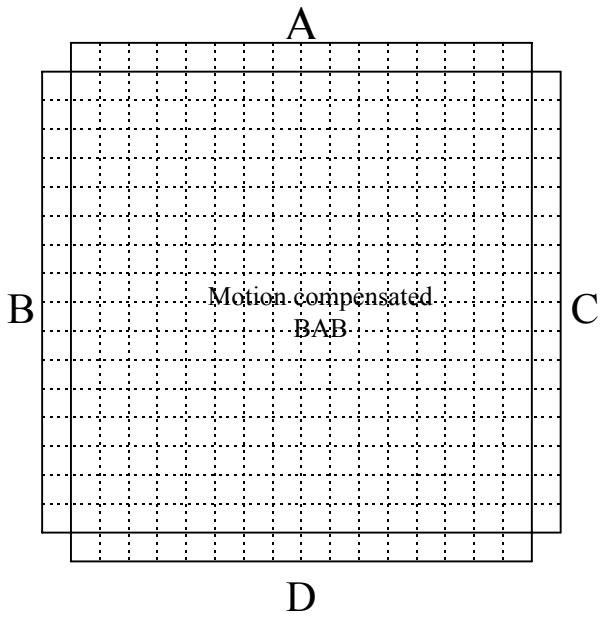


Figure 7-15 — Bordered motion compensated BAB. A: TOP_BORDER. B: LEFT_BORDER. C: RIGHT_BORDER. D: BOTTOM_BORDER

In the case of a motion compensated BAB, the border is also partitioned into 4, as shown Figure 7-15:

- TOP_BORDER, which consists of a line of 16 pixels
- LEFT_BORDER, which consists of a column of 16 pixels
- RIGHT_BORDER, which consists of a column of 16 pixels
- BOTTOM_BORDER, which consists of a line of 16 pixels

The very same sub-sampling process as described above is applied to each of these borders.

7.5.2.5.3 Upsampling

When conv_ratio is different from 1, up-sampling is carried out for the BAB. This is illustrated in Figure 7-16 where “O” in this figure is the coded pixel and “X” is the interpolated pixel. To compute the value of the interpolated pixel, a filter context from the neighboring pixels is first calculated. For the pixel value calculation, the value of “0” is used for a transparent pixel, and “1” for an opaque pixel. The values of the interpolated pixels (Pi, i=1,2,3,4, as shown in Figure 7-17) can then be determined by the following equation:

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

P1 : if(4*A + 2*(B+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P2 : if(4*B + 2*(A+C+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P3 : if(4*C + 2*(B+A+D) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

P4 : if(4*D + 2*(B+C+A) + (E+F+G+H+I+J+K+L) > Th[Cf]) then "1" else "0"

The 8-bit filter context, Cf, is calculated as follows:

$$C_f = \sum_k c_k \cdot 2^k$$

Based on the calculated Cf, the threshold value (Th[Cf]) can be obtained from the look-up table as follows:

Th[256] = {

3, 6, 6, 7, 4, 7, 7, 8, 6, 7, 5, 8, 7, 8, 8, 9,

6, 5, 5, 8, 5, 6, 8, 9, 7, 6, 8, 9, 8, 7, 9, 10,

6, 7, 7, 8, 7, 8, 8, 9, 7, 10, 8, 9, 8, 9, 9, 10,

7, 8, 6, 9, 6, 9, 9, 10, 8, 9, 9, 10, 11, 10, 10, 11,

6, 9, 5, 8, 5, 6, 8, 9, 7, 10, 10, 9, 8, 7, 9, 10,

7, 6, 8, 9, 8, 7, 7, 10, 8, 9, 9, 10, 9, 8, 10, 9,

7, 8, 8, 9, 6, 9, 9, 10, 8, 9, 9, 10, 9, 10, 10, 9,

8, 9, 11, 10, 7, 10, 10, 11, 9, 12, 10, 11, 10, 11, 11, 12,

6, 7, 5, 8, 5, 6, 8, 9, 5, 6, 6, 9, 8, 9, 9, 10,

5, 8, 8, 9, 6, 7, 9, 10, 6, 7, 9, 10, 9, 10, 10, 11,

7, 8, 6, 9, 8, 9, 9, 10, 8, 7, 9, 10, 9, 10, 10, 11,

8, 9, 7, 10, 9, 10, 8, 11, 9, 10, 10, 11, 10, 11, 9, 12,

7, 8, 6, 9, 8, 9, 9, 10, 10, 9, 7, 10, 9, 10, 10, 11,

8, 7, 7, 10, 7, 8, 8, 9, 9, 10, 10, 11, 10, 11, 11, 12,

8, 9, 9, 10, 9, 10, 10, 9, 9, 10, 10, 11, 10, 11, 11, 12,

9, 10, 10, 11, 10, 11, 11, 12, 10, 11, 11, 12, 11, 12, 12, 13 };

TOP_LEFT_BORDER, TOP_RIGHT_BORDER, sub-sampled TOP_BORDER and sub-sampled LEFT_BORDER described in the previous subclause are used. The other pixels outside the BAB are extended from the outermost pixels inside the BAB as shown in Figure 7-16.

In the case that conv_ratio is 4, the interpolation is processed twice. The above mentioned borders of 4x4 BAB are used for the interpolation from 4x4 to 8x8, and top-border (respectively left-border) for the interpolation from 8x8 to 16x16 are up-sampled from the 4x4 BAB top-border (respectively left-border) by simple repetition.

When the BAB is on the left (and/or top) border of VOP, the borders outside VOP are set to zero value. The upsampling filter shall not use pixel values outside of the current video packet.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

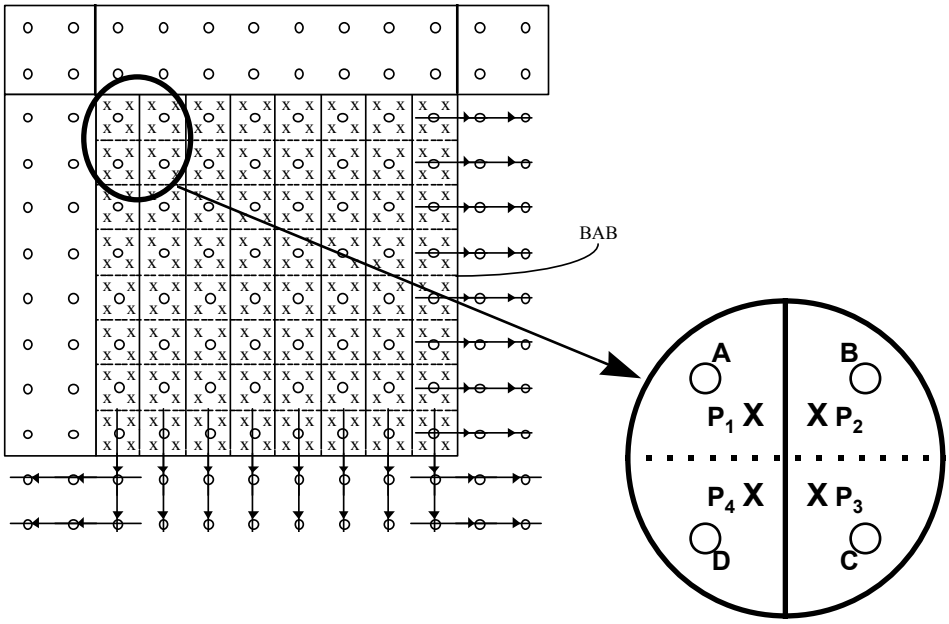


Figure 7-16 — Upsampling

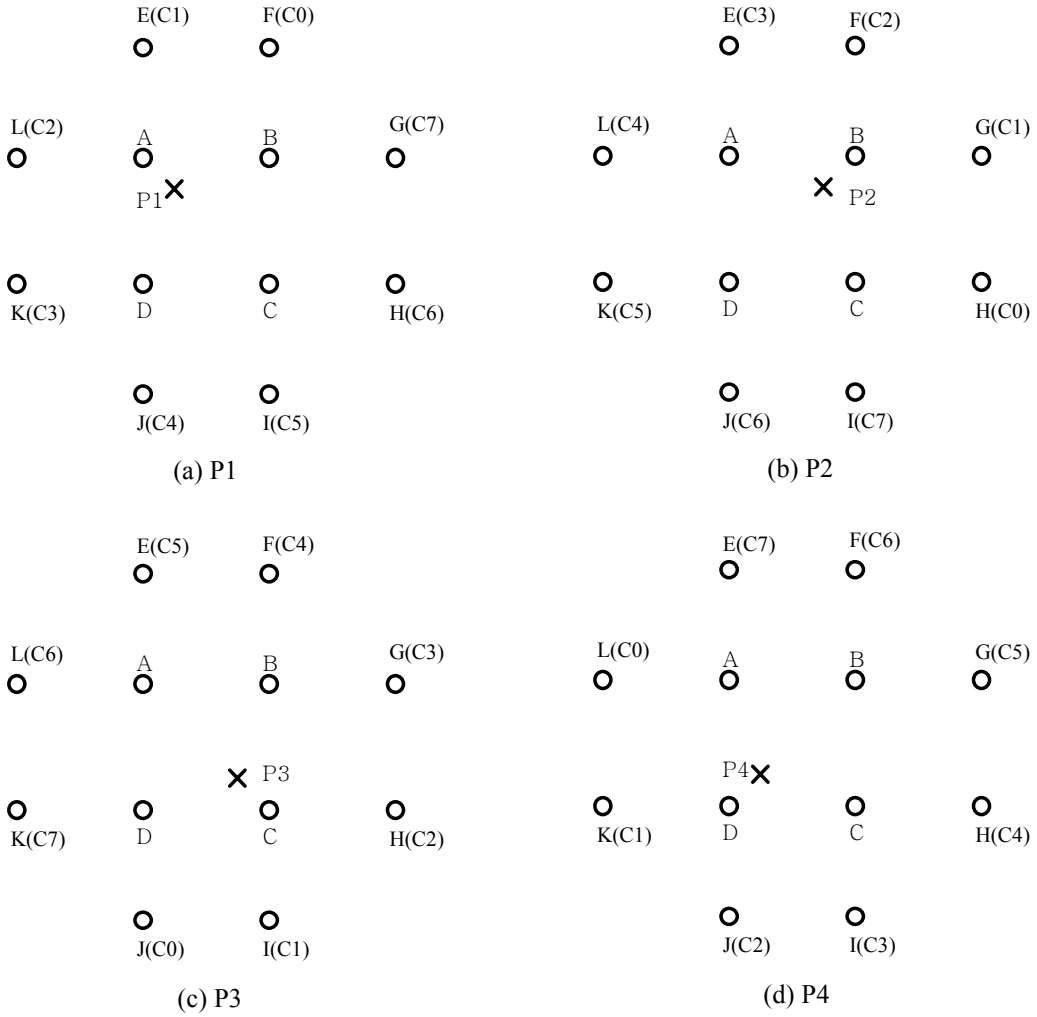


Figure 7-17 — Interpolation filter and interpolation construction

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.2.5.4 Down-sampling process in inter case

If bab_type is '5' or '6' (see Table 7-4), downsampling of the motion compensated bab is needed for calculating the 9 bit context in the case that conv_ratio is not 1. The motion compensated bab of size 16x16 pixels is down sampled to bab of size 16/conv_ratio by 16/conv_ratio pixels by the following rules:

- conv_ratio==2

If the average of pixel values in 2 by 2 pixel block is equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

- conv_ratio==4

If the average of pixel values in 4 by 4 pixel block is equal to or greater than 127.5 the pixel value of the downsampled bab is set to 255 otherwise it is set to 0.

7.5.3 Arithmetic decoding

Arithmetic decoding consists of four main steps:

- Removal of stuffed bits
- Initialization which is performed prior to the decoding of the first symbol
- Decoding of the symbol themselves. The decoding of each symbol may be followed by a re-normalization step.
- Termination which is performed after the decoding of the last symbol

7.5.3.1 Registers, symbols and constants

Several registers, symbols and constants are defined to describe the arithmetic decoder.

- HALF: 32-bit fixed point constant equal to ½ (0x80000000)
- QUARTER: 32-bit fixed point constant equal to ¼ (0x40000000)
- L: 32-bit fixed point register. Contains the lower bound of the interval
- R: 32-bit fixed point register. Contains the range of the interval.
- V: 32-bit fixed point register. Contains the value of the arithmetic code. V is always larger than or equal to L and smaller than L+R.
- p0: 16-bit fixed point register. Probability of the '0' symbol.
- p1: 16-bit fixed point register. Probability of the '1' symbol.
- LPS: boolean. Value of the least probable symbol ('0' or '1').
- bit: boolean. Value of the decoded symbol.
- pLPS: 16-bit fixed point register. Probability of the LPS.
- rLPS: 32-bit fixed point register. Range corresponding to the LPS.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.3.2 Bit stuffing

In order to avoid start code emulation, 1's are stuffed into the bitstream whenever there are too many successive 0's. If the first MAX_HEADING bits are 0's, then a 1 is transmitted after the MAX_HEADING-th 0. If MAX_MIDDLE or more 0's are sent successively a 1 is inserted after the MAX_MIDDLE-th 0. If the number of trailing 0's is larger than MAX_TRAILING, then a 1 is appended to the stream. The decoder shall properly skip these inserted 1's when reading data into the V register (see subclauses 7.5.3.3 and 7.5.3.5).

MAX_HEADING equals 3, MAX_MIDDLE equals 10, and MAX_TRAILIING equals 2.

7.5.3.3 Initialization

The lower bound L is set to 0, the range R to HALF-0x1 (0x7fffffff) and the first 31 bits are read in register V.

7.5.3.4 Decoding a symbol

When decoding a symbol, the probability p0 of the '0' symbol is provided according to the context computed in subclause 7.5.2.5.1 and using Table B.32. p0 uses a 16-bit fixed-point number representation. Since the decoder is binary, the probability of the '1' symbol is defined to be 1 minus the probability of the '0' symbol, i.e. p1 = 1-p0.

The least probable symbol LPS is defined as the symbol with the lowest probability. If both probabilities are equal to ½ (0x8000), the '0' symbol is considered to be the least probable.

The range rLPS associated with the LPS may be computed as R*pLPS: The 16 most significant bits of register R are multiplied by the 16 bits of pLPS to obtain the 32 bit rLPS number.

The interval [L,L+R) is split into two intervals [L,L+R-rLPS) and [L+R-rLPS,L+R). If V is in the latter interval then the decoded symbol is equal to LPS. Otherwise the decoded symbol is the opposite of LPS. The interval [L,L+R) is then reduced to the sub-interval in which V lies.

After the new interval has been computed, the new range R might be smaller than QUARTER. If so, re-normalization is carried out, as described below.

7.5.3.5 Re-normalization

As long as R is smaller than QUARTER, re-normalization is performed.

- f the interval [L,L+R) is within [0,HALF), the interval is scaled to [2L,2L+2R). V is scaled to 2V.
- If the interval [L,L+R) is within [HALF,1) the interval is scaled to [2(L-HALF),2(L-HALF)+2R). V is scaled to 2(V-HALF).
- Otherwise the interval is scaled to [2(L-QUARTER),2(L-QUARTER)+2R). V is scaled to 2(V-QUARTER).

After each scaling, a bit is read and copied into the least significant bit of register V.

7.5.3.6 Termination

After the last symbol has been decoded, additional bits need to be "consumed". They were introduced by the encoder to guarantee decodability.

In general 3 further bits need to be read. However, in some cases, only two bits need to be read. These cases are defined by:

- if the current interval covers entirely [QUARTER-0x1,HALF)
- if the current interval covers entirely [HALF-0x1, 3QUARTER)

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

After these additional bits have been read, 32 bits shall be “unread”, i.e. put the content of register V back into the bit buffer.

7.5.3.7 Software

The example software for arithmetic decoding for binary shape decoding is included in Annex B.

7.5.4 Spatial scalable binary shape decoding

7.5.4.1 Decoding of base layer

The decoding process of the base layer is the same as non-scalable binary shape decoding process.

7.5.4.2 Decoding of enhancement layer

When spatial scalability is enabled (scalability is set to 1 and hierarchy_type is set to 0) with enhancement_type == 0 or when spatial scalability is enabled with enhancement_type == 1 and use_ref_shape == 0, scalable shape coding process is used for decoding of binary shape.

If spatial scalability is enabled, use_ref_shape is set to 1 and enhancement_type is set to 1, the same non-scalable decoding process is applied for binary shape of enhancement layer. In this case, the following rules are applied for enhancement layer.

1. In PVOP, Inter shape coding should be done as bab_type of co-located MB in the reference VOP (lower layer) is “Opaque”.

2. In BVOP, forward reference VOP, most recently decoded non-empty VOP in the same layer, is always selected as reference VOP of shape coding.

If spatial scalability is enabled, use_ref_shape is set to 1 and enhancement_type is set to 0, then the up-sampled binary shape from reference layer is used for the binary shape of enhancement layer. The up sampling and down sampling process of this purpose also follows up-down sampling method described in the subclause 7.5.4.4.

When spatial scalability is enabled and enhancement_type is set to 0, in the enhancement layer, the forward prediction in P-VOP and the backward prediction in B-VOP are used as the spatial prediction. In that case the shape information is coded by scan interleaving (SI) based method. For the forward prediction in B-VOP a motion compensated temporal prediction is made from the reference VOP in the enhancement layer. In that case the shape information is coded by the CAE method as like in base layer except that the shape motion vectors are obtained from those of the collocated bab in the lower layer. Motion vector and shape coding mode(bab_type) of collocated bab in the lower layer are used for decoding the enhancement layer bab.

The location of collocated bab in the lower layer can be found by following method.

```
collocated_MBX
    = min ( max ( 0, current_MBX*shape_hor_sampling_factor_m/shape_hor_sampling_factor_n ),
            NumMBXLower-1 );

collocated_MBY
    = min ( max( 0, current_MBY*shape_ver_sampling_factor_m/shap_ever_sampling_factor_n ),
            NumMBYLower-1);
```

For the current MB location [current_MBX, current_MBY], the location of collocated bab in the reference layer is denoted as [collocated_MBX, collocated_MBY]. current_MBX, current_MBY, collocated_MBX and collocated_MBY are the MB-unit coordinations. NumMBXLower and NumMBYLower denote the number of microblocks in the lower layer VOP on horizontal and vertical directions, respectively.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.4.3 Prediction in the enhancement layer

A motion compensated temporal prediction is made from the reference VOP in the enhancement layer. In addition, a spatial prediction is formed from the lower reference layer decoded VOP. These predictions are selected individually or combined to form the actual prediction.

vop_shape_coding_type is not used in the Spatial Scalable Binary Shape Coding process.

7.5.4.4 Spatial prediction

The spatial prediction is made by resampling the lower reference layer reconstructed VOP to the same sampling grid as the enhancement layer. For the resampling, repetition is used on the lower layer.

For enhancement layer encoding/decoding, the reference layer VOP should be up-sampled as the sampling ratio, which is included in the VOL syntax. In VOL syntax for enhancement layer, there are four fields for the up-sampling ratio, i.e., shape_hor_sampling_factor_n, shape_hor_sampling_factor_m, shape_vert_sampling_factor_n and shape_vert_sampling_factor_n.

Down sampling
For the sampling rate, below sampling process is adopted for both horizontal and vertical directions.

For the down-sampling rate m/n,

$m/n = [1/2^k][(m2^k)/n]$, where k is the largest integer that is equal or smaller than $\log_2(n/m)$.

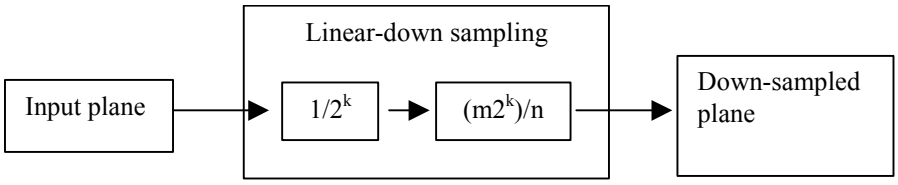


Figure 7-18 — Down sampling process

1. Find $k = \text{floor}(\log_2(n/m))$.

2. Sampling a pixel per 2^k pixels:

   if($x\%2^k == (2^k - 1)$) Sampling F(x) pixel;

3. On the result of 2, sampling a pixel per $n/(m2^k)$ pixels.

Up sampling
For the sampling rate, below sampling process is adopted for both horizontal and vertical directions.

For the up-sampling rate n/m,

$n/m = [n/(m2^k)]\ [2^k]$, where k is the largest integer that is equal or smaller than $\log_2(n/m)$.

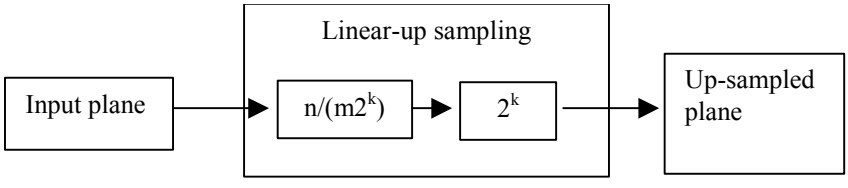


Figure 7-19 — Up sampling process

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1. Find k = floor($\log_2(n/m)$).

2. Repeat one pixel per $(m2^k)/(n-m2^k)$ pixels.

3. On the result of 2, Repeat $2^k - 1$ pixels per pixel.

For a case that the sampling ratio is 1/2 for both horizontal and vertical direction, one pixel value in the lower layer is upsampled to 2x2 pixels in the current layer by the following methods:

The binary alpha image $f_{current}[y_H][x_H]$ in the enhancement layer is upsampled by repetition of the pixels of the image $f_{lower}[y_L][x_L]$ in the lower layer according to the following formula:

$$f_{current}[2\,y_L+i][2\,x_L+j] = f_{lower}[y_L][x_L], \qquad (i, j=0, 1)$$

In Figure 7-20, the spatial prediction of one BAB is shown for example. In the enhancement layer the pixel value $f_{current}[2y_L+1][2x_L+1$ is always same with the corresponding pixel value $f_{lower}[y_L][x_L]$ in the lower layer.

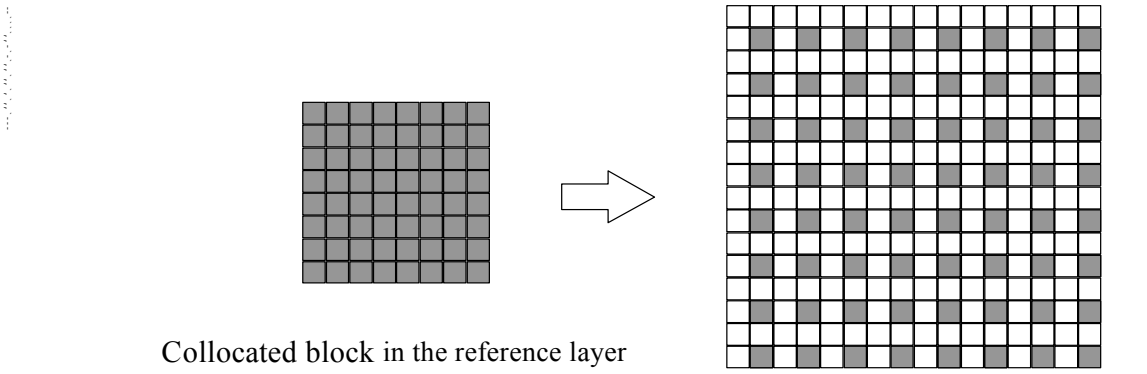


Figure 7-20 — Associated pixel locations and spatial prediction of current BAB by repetition

In the case of arbitrary shape VOPs, its size and locations change from time to time. In order to ensure the formation of the spatial prediction, it is necessary to identify the location of the resampled VOPs in the absolute coordinate system. vop_horizontal_mc_spatial_ref and vop_vertical_mc_spatial_ref in the resampled reference layer should be coincident with those in the enhancment layer. Hence after the position of the spatial reference of the reference layer is scaled by *_sampling_factor_n/*_sampling_factor_m, the resampled reference VOP should be relocated by referencing the offset values that are the differences between the positions of the spatial references of two VOPs in the horizontal and the vertical direction.

Since the shape motion vectors and the shape modes in the reference layer are also needed for the actual shape coding in the enhancement layer bab, the bab type matrix and the motion vector matrix are resampled by repetition. In case of the motion vector matrix each components are scaled by the sampling ratio on both directions.

7.5.4.5 Temporal prediction

A motion compensated temporal prediction is made from the reference VOP in the enhancement layer. For predicting a bab, the motion vectors of the collocated bab in the lower layer is used. Enhancement layer shape motion vector [mvX, mvY] can be used by scaling the lower layer shape motion vector [mvXLower, mvYLower]. The scaling method is as follows:

mvX = (mvXLower * shape_hor_sampling_factor_n) // shape_hor_sampling_factor_m

mvY = (mvYLower * shape_ver_sampling_factor_n) // shape_ver_sampling_factor_m

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.4.6 Types for enhancement layer bab

Each bab belongs to one of four types given in Table 7-6. This enh_bab_type field influences decoding of further shape information of enhancement layers. For P-VOP coding only 0 and 1 of the enh_bab_type are allowed. For B-VOP coding scheme, all of the four modes are allowed.

Table 7-6 — List of enh_bab_types and usage

enh_bab_type	Semantic	Used in
0	intra NOT_CODED	P-/B- VOPs
1	intra CODED	P-/B- VOPs
2	inter NOT_CODED	B- VOPs
3	inter CODED	B- VOPs

When the enh_bab_type == 1, the bab is predicted by upsampling from the collocated block in the lower reference layer and is decoded by intra-mode CAE based on Scan Interleaving(SI).

The enh_bab_type == 0 means that the error of the predicted bab from the lower reference layer is less than the threshold. The predicted bab is given by upsampling the collocated block in the lower reference layer. The spatial prediction method in 7.5.4.4 or linear repetition method can be used for the upsampling. The predicted bab is used as the current bab.

For enh_bab_type==2 and enh_bab_type==3, no motion vector is decoded for the temporal prediction of enhancement layer bab. The motion vector of the collocated block in the lower reference layer can be utilized. Each component of the vector are scaled by two for referencing. If the collocated bab in the lower reference layer is intra coding mode, the value of the motion vector is set to zero. If enh_bab_type is decode as "inter NOT_CODED", then the error of the predicted bab from the previous VOP is less than the threshold. And the predicted bab is used as the current bab. If enh_bab_type is decode as "inter CODED", the current bab is decoded by inter-CAE which is used in non-scalable shape decoding.

7.5.4.7 Decoding bab types for enhancement layer

In order to decode the bab types, the collocated bab of the lower layer is also examined. Based on the lower layer bab types, the following VLC table is used for decoding the enh_bab_type field. For B-VOP decoding in enhancement layer enh_bab_type is decoded using in Table 7-8. For P-VOP decoding in enhancement layer, there are only two enh_bab_type fields, i.e., 0 and 1, and they are decoded by 1~2 bit described in Table 7-7.

Table 7-7 — Decoding of enh_bab_type for P-VOP in enhancement layer

enh_bab_type	Code
0	1
1	01

Table 7-8 — Decoding of enh_bab_type for B-VOP in enhancement layer

		enh_bab_type in the enhancement layer			
		0	**1**	**2**	**3**
enh_bab_type in the lower reference layer or (bab_type in the base layer)	**0(2,3)**	1	01	001	000
	1(4)	110	0	10	111
	2(0,1)	001	01	1	000
	3(5,6)	110	0	111	10

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note that the numbers 0-3 in these tables correspond to the enh_bab_types given in 7.8.2.1.6. When the lower layer is the base layer, {"MVDs==0 && No Update", "MVDs!=0 && No Update"}, {"all_0", "all_255"}, {"intraCAE"} and {"MVDs==0 && interCAE", "MVDs!=0 && interCAE"} modes are converted to the enh_bab_type 2, 0, 1 and 3, respectively.

7.5.4.8 Intra coded enhancement layer decoding

Intra coded enhancement layer decoding uses scan interleaving algorithm before performing intra-mode CAE. The decoding order with SI scanning is as follows:

1. Copy B from reference layer

2. Decoding order with Vertical scanning : Vr --> Vp1 --> Vp2 --> ... --> Vpk

3. Decoding order with Horizontal scanning : Hr --> Hp1 --> Hp2 --> ... --> Hpl

where,

B: Pixel that can be copied from collocated pixel in the reference layer.V_r: Vertical scanning Pixel of Residual term of vertical linear-up sampling

V_{pk}: Vertical scanning Pixel of 2^k term of vertical linear-up sampling

H_r: Horizontal scanning Pixel of Residual term of Horizontal linear-up sampling

H_{pl}: Horizontal scanning Pixel of 2^l term of Horizontal linear-up sampling

k: The largest integer that is equal or smaller than $\log_2$(shape_hor_sampling_factor_n / shape_hor_sampling_factor_m)

l: The largest integer that is equal or smaller than $\log_2$(shape_ver_sampling_factor_n / shape_ver_sampling_factor_m)

For an example, when shape_hor_sampling_factor_n = 8, shape_hor_sampling_factor_m = 3, shape_ver_sampling_factor_n = 5, shape_hor_sampling_factor_m =1, the scan order of pixels for decoding is shown in Figure 7-21.

--`,,`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr	Hr
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Vp1	Vr	Vp1	B	Vp1	B	Vp1	B	Vp1	Vr	Vp1	B	Vp1	B	Vp1	B
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Vp1	Vr	Vp1	B	Vp1	B	Vp1	B	Vp1	Vr	Vp1	B	Vp1	B	Vp1	B
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1	Hp1
Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2	Hp2
Vp1	Vr	Vp1	B	Vp1	B	Vp1	B	Vp1	Vr	Vp1	B	Vp1	B	Vp1	B

Figure 7-21 — An example of decoding order when enh_bab_type==1 (Decoding order: B → V_r → V_{p1} → H_r → H_{p1} → H_{p2})

7.5.4.9 Border formation for intra-mode CAE for enhancement layer decoding

The border formation for decoding of intra CODED bab (enh_bab_type==1) is the same as that of the non-scalable shape coding (see subclause 7.5.2.5.2) except E and U regions in Figure 7-14. The border of E and U region of the current bordered bab contains the pixels from the collocated regions in the up-sampled lower layer plane and which consists of 2 lines in bottom and right of the current bab.

7.5.4.10 Scan Interleaving Algorithm

The spatial (or resolution) scalability can be easily sought by incorporating the so-called Scan Interleaving (SI) method. In SI, coded-scan-lines are decoded by using the prediction from the closest upper and lower neighboring two lines (reference-scan-lines) as shown in Figure 7-22. In the figure, horizontal direction scanning is described. The pixels on coded-scan-lines are the pixels to be decoded and the pixels on reference-scan-lines are the pixels that are already reconstructed The closest upper and lower neighboring pixel values in reference-scan-lines are used for reconstructing the current pixel. (The closest upper and lower neighboring pixels mean the closest left and right neighboring pixels in vertical direction scanning case.) By using those two reference pixels, the current decoded pixel has three types as follows:

Transitional sample data (TSD): The closest upper and lower neighboring pixels that are already reconstructed have different colors each other.

Exceptional sample data (ESD): The closest upper and lower neighboring pixels that are already reconstructed are same and the current pixel is different with the neighboring pixels.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Predictable sample data (PSD): The closest upper and lower neighboring pixels that are already reconstructed are the same as the current pixel.

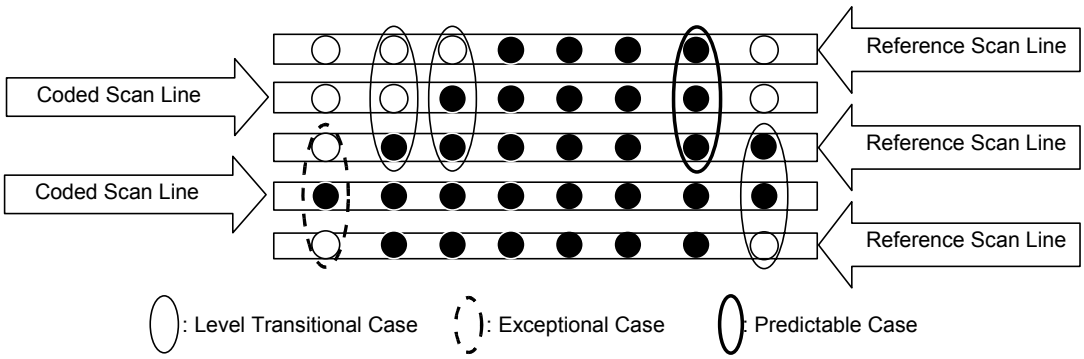


Figure 7-22 — Scan Interleaving for Scalable Coding: an example of a times by 2 case

To encoding/decoding shape mask plane, SI performs the scanning method in vertical and horizontal directions, successively. SI with vertical scanning (V-SI) and SI with horizontal scanning (H-SI) are separately applied to the input image.

For decoding a bab in the current layer, the spatially predicted bab from collocated block in the lower layer is enhanced by V-SI. And then H-SI is successively performed on the enhanced plane of the V-SI. The reconstructing order is presented in subclause 7.5.4.8.

In the first step of decoding a bab, a binary flag, which denotes the current bab type of SI, is decoded by BAC. There are two bab types of SI : transitional bab (TSD-B) and exceptional bab (ESD-B). If a bab has no ESD, the bab is set to TSD-B. Otherwise the bab is set to ESD-B. When the flag is '0', the current bab is TSD-B. Otherwise, the current bab is ESD-B.

In the encoder, for encoding TSD-B, only TSD pixel are encoded by CAE with the context shown in Figure 7-24. And for encoding ESD-B, all the pixels in the coded-scan-lines are encoded by CAE with the context shown in Figure 7-24. The encoding order is the same as shown in subclause 7.5.4.8.

If a bab is TSD-B, only TSD are decoded in the order of V-SI and H-SI. The specific decoding order is presented in 7.5.4.8. The decoding process of TSD-B is as follows.

Following the scanning order, if the current pixel is TSD, then the pixel is decoded by BAC with the context in Figure 7-24. And if the two neighbor pixels (the closest upper(left) and bottom(right)) are the same value, then the current pixel has the same value as its neighbor pixels.

If a bab is ESD-B, all the pixels in coded scan line are decoded by BAC with the context in Figure 7-24.

7.5.4.11 Determination of the scanning type in enhancement layer decoding

When enh_bab_type is 1 (intra CODED) and the scale ratio between the layers is square (i.e 200x300 <--> 400x600), before performing BAC, the scanning type should be determined. For the determination, the top and left borders of the collocated block of the lower layer are extended with its neighboring pixels like Figure 7-23. If an extended border is outside VOP, the pixel values of the extended border are set to zero. Two counts, N_L and N_U are calculated (N_L: the number of pixels which have different values from the left neighboring pixel value; N_U: the number of pixels which have different values from the upper neighboring value). If N_U is greater than N_L, the spatially predicted bab for decoding the enhancement layer bab has to be transposed and then the intra-mode CAE for enhancement layer is performed, Otherwise, the intra-mode CAE for enhancement layer is performed on the spatially predicted bab without transposing, that is default condition.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

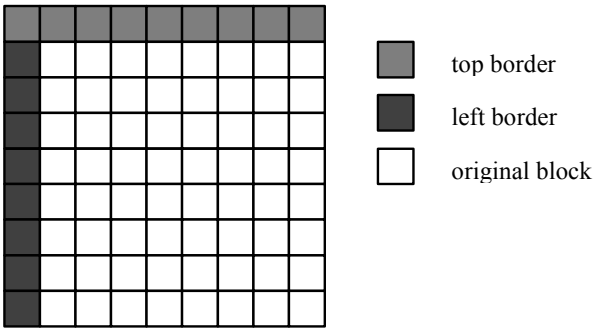


Figure 7-23 — Extension of border pixels

7.5.4.12 Decoding of bab type of SI

The first decoded symbol of enh_binary_arithmetic_code(), which performs BAC, denotes the current bab type of SI. The probability of SI_bab_type_prob[] given in Annex B is used for decoding bab type of SI. When this flag is '0', the current bab is TSD-B. Otherwise, the current bab is ESD-B.

7.5.4.13 Decoding of intra-mode CAE for enhancement layer

For decoding intra mode CAE for enhancement layer, BAC using the V-SI and H-SI decoding method is performed. By using neighboring 7 bits context, a pixel is arithmetic decoded as in CAE, such that:

Compute a context using the templates shown in Figure 7-24 ((a): for V-SI, (b): for H-SI).

Using the context, find probability from the probability table, i.e. enh_intra_v_prob[] for V-SI and enh_intra_h_prob[] for H-SI. These tables are given in Annex B.

Decoding the pixel value using the probability.

The following subclause describes the computation of the contexts for intra mode CAE.

7.5.4.14 Contexts for intra-mode CAE for enhancement layer

Depending on the scanning direction, a 7 bit context $C = \sum_k c_k \cdot 2^k$ is built for each coded pixel as shown in Figure 7-24, where c_k=0 for transparent pixels and c_k=1 for opaque pixels. (a) of Figure 7-24 is for the intra-mode CAE with V-SI (SI by the vertical scanning), and (b) of Figure 7-24 is for the intra-mode CAE with H-SI (SI by the horizontal scanning). In (a) of the figure, C6, C5 and C4 are the closest decoded pixels in top-left, top, and top-right location, respectively. C3, and C2 are the closest decoded pixels in left and right location, respectively. And C1 and C0 are the closest decoded pixels in bottom-left and bottom-right location, respectively.

In (b) of the figure, C6, C5 and C4 are the closest decoded pixels in top-left, top, and top-right location, respectively. C3 is the closest decoded pixel in left location. And C2, C1 and C0 are the closest decoded pixels in bottom-left, bottom, and bottom-right location, respectively.

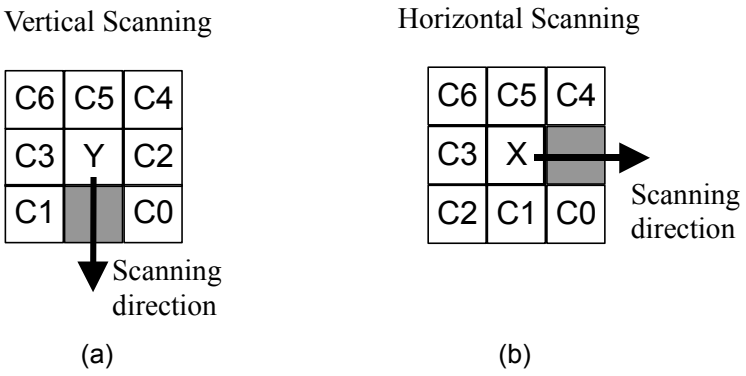


Figure 7-24 — The context information at different scanning of SI

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

There are some special cases to consider:

When building contexts, any pixels outside the bounding box of the current VOP to the left and above are assumed to be zero (transparent).

When building contexts and resync_marker_disabled==0, any pixels outside the space of the current video packet to the left and above are assumed to be zero (transparent).

When constructing the context, any pixel outside the bordered BAB is assumed to be the same pixel value of the closest pixel in the border of the current BAB from the context location.

7.5.4.15 Decoding of Inter-mode CAE for enhancement layer

For decoding of enhancement layer with B-VOP coding, the inter-mode CAE is performed in the case of the “inter coded” mode (enh_bab_type==3). The context information in Figure 7-25 is utilized for this decoding. The decoding process of a pixel is done in the same way as non-scalable binary shape coding.

When constructing the Inter-mode CAE context the following conditional assignment is performed.

if(c1 is unknown) c1=c2

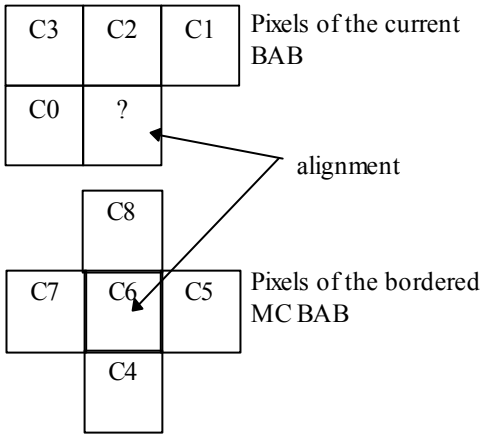


Figure 7-25 — The templates of Inter-mode CAE for enhancement layer. C6 is aligned with the pixel to be decoded. The pixel to be decoded is marked with ‘?’

7.5.4.16 Quality (SNR) Scalable Coding

Quality Scalability can be easily sought by spatial scalability. Using the spatial scalability and spatial prediction, binary shape can achieve the quality (SNR) scalability. In the decoding process, each spatial scalable layer can be spatially predicted to the full resolution. For the binary shape, this scalable structure is equivalent to quality scalable structure.

7.5.5 Grayscale Shape Decoding

Grayscale alpha plane decoding is achieved by the separate decoding of a support region and the values of the alpha channel. The support region is transmitted by using the binary shape as described above. The alpha values are transmitted as texture data with arbitrary shape, using almost the same coding method as is used for the luminance texture channel.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

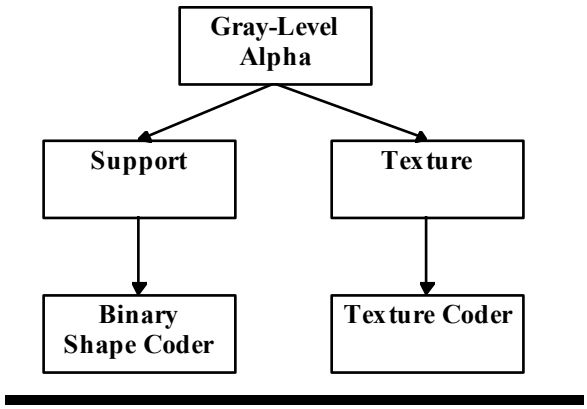


Figure 7-26 — Grayscale shape coding

All samples which are indicated to be transparent by the binary shape data, must be set to zero in the decoded grayscale alpha plane. Within the VOP, alpha samples have the values produced by the grayscale alpha decoding process. Decoding of binary shape information is not dependent on the decoding of grayscale alpha. The alpha values are decoded into 16x16 macroblocks in the same way as the luminance channel (see subclauses 7.4 and 7.6, and 7.8.7). The 16x16 blocks of alpha values are referred to as alpha macroblocks hereafter. The data for each alpha macroblock is present in the bitstream immediately following the texture data for the corresponding texture macroblock. Any aspect of alpha decoding that is not covered in this document should be assumed to be the same as for the decoding of luminance.

7.5.5.1 Grayscale Alpha COD Modes

When decoding grayscale alpha macroblocks, CODA is first encountered and indicates the coding status for alpha. It is important to understand that the macroblock syntax elements for alpha are still present in the bitstream for P, B, or S(GMC) macroblocks even if the texture syntax elements indicate “not-coded” (not_coded=’1’). In this respect, the decoding of the alpha and texture data are independent. The only exception is for B-VOPs when the colocated P-VOP texture macroblock is skipped. In this case, no syntax is transmitted for texture or grayscale alpha, as both types of macroblock are skipped.

For macroblocks which are completely transparent (indicated by the binary shape coding), no alpha syntax elements are present and the grayscale alpha samples must all be set to zero (transparent). If CODA=”all opaque” (I, P, B, or S(GMC) macroblocks) or CODA=”not coded” (P, B, or S(GMC) macroblocks) then no more alpha data is present. Otherwise, other alpha syntax elements follow, including the coded block pattern (cbpa), followed by alpha texture data for those 8x8 blocks which are coded and non-transparent, as is the case for regular luminance macroblock texture data.

When CODA=”all opaque”, the corresponding decoded alpha macroblock is filled with a constant value of 255. This value will be called AlphaOpaqueValue.

7.5.5.2 Alpha Plane Scale Factor

For both binary and grayscale shape, the VOP header syntax element “vop_constant_alpha” can be used to scale the alpha plane. If this bit is equal to ‘1’, then each pixel in the decoded VOP is scaled before output, using vop_constant_alpha_value. The scaling formula is:

scaled_pixel = (original_pixel * (vop_constant_alpha_value + 1)) / 256

Scaling is applied at the output of the decoder, such that the decoded original values, not the scaled values are used as the source for motion compensation.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.5.3 Gray Scale Quantiser Scale

When no_gray_quant_update is equal to “1”, the grayscale alpha quantiser scale is fixed for all macroblocks to the value indicated by vop_alpha_quant. Otherwise, the grayscale quantiser scale is reset at each new macroblock to a value that depends on the current texture quantiser scale (after any update by dquant). The relation is:

current_alpha_quant = (current_texture_quant * vop_alpha_quant) / vop_quant

The resulting value of current_alpha_quant must then be clipped so that it never becomes less than 1.

7.5.5.4 Intra Macroblocks

When the texture mb_type indicates an intra macroblock in an I-, P-, or S(GMC)-VOP , the grayscale alpha data is also decoded using intra mode.

The intra DC value is decoded in the same way as for luminance, using the same non-linear transform to convert from alpha_quant to DCScalarA. However, intra_dc_vlc_thr is not used for alpha, and therefore AC coeffiecient VLCs are never used to code the differential intra DC coefficient.

DC prediction is used in the same way as for luminance. However, when coda_i indicates that a macroblock is all opaque, a synthetic intra DC value is created for each block in the current macroblock so that adjacent macroblocks can correctly obtain intra DC prediction values. The synthetic intra DC value is given as:

BlockIntraDC = (((AlphaOpaqueValue * 8) + (DcScalerA>>1)) / DcScalerA) * DcScalerA

AlphaOpaqueValue is described in subclause 7.5.4.1.

The intra cbpa VLC makes use of the *inter* cbpy VLC table, but the intra alpha block DCT coefficients are decoded in the same manner as with luminance intra macroblocks.

When interlaced is equal to “1”, alternate scan should not be applied to coding of gray-level alpha.

When both interlaced is equal to “1” and video_object_layer_shape is equal to “11” (grayscale), only the frame DCT should be applied to coding of gray-level alpha.

7.5.5.5 Inter Macroblocks and Motion Compensation

Motion compensation is carried out for P-, B-, and S(GMC)-VOPs, using the 8x8 or 16x16 luminance motion vectors, or the warping of the previous decoded VOP, in the same way as for luminance data, except that regular motion compensation is used instead of OBMC. Forward, backward, bidirectional and direct mode motion compensation are used for B-VOPs. Where the luminance motion vectors are not present because the texture macroblock is skipped, the exact same style of non-coded motion compensation used for luminance is applied to the alpha data (but without OBMC). Note that this does not imply that the alpha macroblock is skipped, because an error signal to update the resulting motion compensated alpha macroblock may still be present if indicated by coda_pb. When the colocated P-VOP texture macroblock is skipped for B-VOPs, then the alpha macroblock is assumed to be skipped with no syntax transmitted.

cbpa and the alpha inter DCT coefficients are decoded in the same way as with luminance cbpy and inter DCT cofficients.

When both interlaced is equal to “1” and video_object_layer_shape is equal to “11” (grayscale), the field padding should be applied to coding of gray-level alpha.

When both interlaced is equal to “1” and video_object_layer_shape is equal to “11” (grayscale), both frame and field motion compensation are applicable in coding of texture and gray-level alpha, but only the frame DCT should be applied to coding of gray-level alpha.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.5.6 Method to be used when blending with greyscale alpha signal

The following explains the blending method to be applied to the video object in the compositor, which is controlled by the composition_method flag and the linear_composition flag. The linear_composition flag is informative only, and the decoder may ignore it and proceed as if it had the value 0. However, it is normative that the composition_method flag be acted upon.

The descriptions below show the processing taking place in YUV space; note that the processing can of course be implemented in RGB space to obtain equivalent results.

composition_method=0 (cross-fading)

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

$$Pyuv = ((2^n-1 - Nalpha) * Myuv + (Nalpha * Nyuv)) / (2^n-1)$$

$$Palpha = (2^n-1)$$

composition_method=1 (Additive mixing)

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

$$Pyuv = \begin{cases} Myuv & ..... Nalpha = 0 \\ \{ (Myuv - BLACK) - ((Myuv - BLACK) * Nalpha) / (2^n-1) + Nyuv & ..... Nalpha > 0 \end{cases}$$

(this is equivalent to Pyuv = Myuv*(1-alpha) + Nyuv, taking account of black level and the fact that the video decoder does not produce an output in areas where alpha=0)

$$Palpha = Nalpha + Malpha - (Nalpha*Malpha) / (2^n-1)$$

where

BLACK is the common black value of foreground and background objects.

NOTE The compositor must convert foreground and background objects to the same black value and signal range before composition. The black level of each video object is specified by the video_range bit in the video_signal_type field, or by the default value if the field is not present. (The RGB values of synthetic objects are specified in a range from 0 to 1, as described in ISO/IEC 14496-1:2001).

- linear_composition = 0: The compositing process is carried out using the video signal in the format from which it is produced by the video decoder, that is, without converting to linear signals. Note that because video signals are usually non-linear ("gamma-corrected"), the composition will be approximate.

- linear_composition = 1: The compositing process is carried out using linear signals, so the output of the video decoder is converted to linear if it was originally in a non-linear form, as specified by the video_signal_type field. Note that the alpha signal is always linear, and therefore requires no conversion.

7.5.6 Multiple Auxiliary Component Decoding

Auxiliary components are defined for the VOP on a pixel-by-pixel basis, and contain data related to the video object, like disparity, depth, additional texture. Up to 3 auxiliary components (including the grayscale shape) are possible. The number and type of these components is indicated by the video_object_layer_shape_extension given in Table 6-17. For example, a value '0000' indicates the grayscale (alpha) shape. The same support region as described in 7.5.4 is used for all auxiliary components, and the decoding procedure is the same as described in subclauses 7.5.4.1 - 7.5.4.6.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6 Motion compensation decoding

In order to perform motion compensated prediction on a per VOP basis, a special padding technique, i.e. the macroblock-based repetitive padding, is applied for the reference VOP. The details of these techniques are described in the following subclauses.

Since a VOP may have arbitrary shape, and this shape can change from one instance to another, conventions are necessary to ensure the consistency of the motion compensation process.

The absolute (frame) coordinate system is used for referencing every VOP. At every given instance, a bounding rectangle that includes the shape of that VOP, as described in subclause 7.5, is defined. The left and top corner, in the absolute coordinates, of the bounding rectangle is decoded from VOP spatial reference. Thus, the motion vector for a particular feature inside a VOP, e.g. a macroblock, refers to the displacement of the feature in absolute coordinates. No alignment of VOP bounding rectangles at different time instances is performed.

For motion compensation, the amplitude resolution of the motion vectors is signaled by the quarter_sample flag in the VOL header, see subclause 6.2.3. Either ½ pel resolution ("half sample mode", quarter_sample==0) or ¼ pel ("quarter sample mode", quarter_sample==1) is used. For ½ pel resolution, the necessary ½ pel samples are calculated by means of half sample interpolation as described in subclause 7.6.2.1. For ¼ pel resolution, the necessary ½ and ¼ pel samples are calculated by means of half and quarter sample interpolation as described in subclause 7.6.2.2.

In addition to the above motion compensation processing, three additional processes are supported, namely, unrestricted motion compensation, four MV motion compensation,and overlapped motion compensation. Note that in all three modes, macroblock-based padding of the arbitrarily shaped reference VOP is performed for motion compensation.

7.6.1 Padding process

The padding process defines the values of luminance and chrominance samples outside the VOP for prediction of arbitrarily shaped objects. Figure 7-27 shows a simplified diagram of this process.

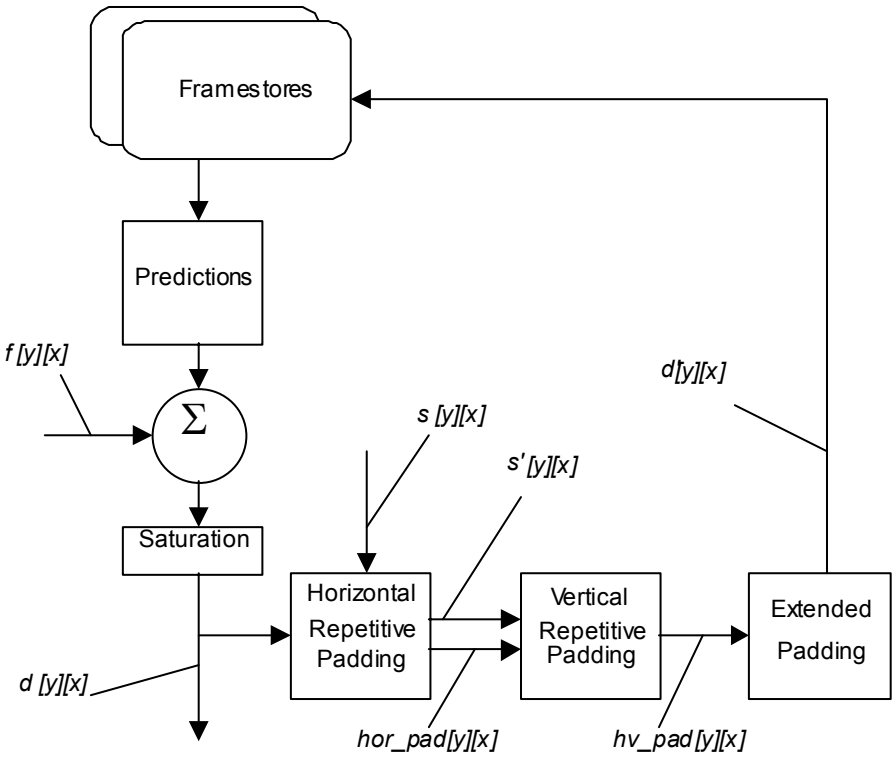


Figure 7-27 — Simplified padding process

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A decoded macroblock *d[y][x]* is padded by referring to the corresponding decoded shape block *s[y][x]*. The luminance component is padded per 16 x 16 samples, while the chrominance components are padded per 8 x 8 samples. A macroblock that lies on the VOP boundary (hereafter referred to as a boundary macroblock) is padded by replicating the boundary samples of the VOP towards the exterior. This process is divided into horizontal repetitive padding and vertical repetitive padding. The remaining macroblocks that are completely outside the VOP (hereafter referred to as exterior macroblocks) are filled by extended padding.

NOTE The padding process is applied to all macroblocks inside the bounding rectangle of a VOP. The bounding rectangle of the luminance component is defined by vop_width and vop_height extended to multiple of 16, while that of the chrominance components is defined by (vop_width>>1) and (vop_height>>1) extended to multiple of 8.

7.6.1.1 Horizontal repetitive padding

Each sample at the boundary of a VOP is replicated horizontally to the left and/or right direction in order to fill the transparent region outside the VOP of a boundary macroblock. If there are two boundary sample values for filling a sample outside of a VOP, the two boundary samples are averaged (//2).

hor_pad[y][x] is generated by any process equivalent to the following example. For every line with at least one shape sample *s[y][x]* == 1(inside the VOP):

```
for (x=0; x<N; x++) {
    if (s[y][x] == 1) { hor_pad[y][x] = d[y][x]; s'[y][x] = 1; }
    else {
        if ( s[y][x'] == 1 && s[y][x"] == 1)  {
            hor_pad[y][x] = (d[y][x']+ d[y][x"])//2;
            s'[y][x] = 1;
        } else if ( s[y][x'] == 1 ) {
            hor_pad[y][x] = d[y][x']; s'[y][x] = 1;
        } else if ( s[y][x"] == 1 ) {
            hor_pad[y][x] = d[y][x"]; s'[y][x] = 1;
        }
    }
}
```

where *x'* is the location of the nearest valid sample (*s[y][x']* == 1) at the VOP boundary to the left of the current location *x*, *x"* is the location of the nearest boundary sample to the right, and *N* is the number of samples of a line in a macroblock. *s'[y][x]* is initialized to 0.

7.6.1.2 Vertical repetitive padding

The remaining unfilled transparent horizontal samples (where *s'[y][x]* == 0) from subclause 7.6.1.1 are padded by a similar process as the horizontal repetitive padding but in the *vertical* direction. The samples already filled in subclause 7.6.1.1 are treated as if they were inside the VOP for the purpose of this vertical pass.

hv_pad[y][x] is generated by any process equivalent to the following example. For every column of *hor_pad[y][x]* :

```
for (y=0; y<M; y++) {
    if (s'[y][x] == 1)
        hv_pad[y][x] =hor_pad[y][x];
    else {
        if ( s'[y'][x] == 1 && s'[y"][x] == 1 )
            hv_pad[y][x] = (hor_pad[y'][x] + hor_pad[y"][x])//2;
        else if ( s'[y'][x] == 1 )
            hv_pad[y][x] = hor_pad[y'][x];
        else if (s'[y"][x] == 1 )
            hv_pad[y][x] = hor_pad[y"][x];
    }
}
```

where *y'* is the location of the nearest valid sample (*s'[y'][x]* == 1) above the current location *y* at the boundary of *hv_pad*, *y"* is the location of the nearest boundary sample below *y*, and *M* is the number of samples of a column in a macroblock.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.1.3 Extended padding

Exterior macroblocks immediately next to boundary macroblocks are filled by replicating the samples at the border of the boundary macroblocks. Note that the boundary macroblocks have been completely padded in subclause 7.6.1.1 and subclause 7.6.1.2. If an exterior macroblock is next to more than one boundary macroblocks, one of the macroblocks is chosen, according to the following convention, for reference.

The boundary macroblocks surrounding an exterior macroblock are numbered in priority according to Figure 7-28. The exterior macroblock is then padded by replicating upwards, downwards, leftwards, or rightwards the row of samples from the horizontal or vertical border of the boundary macroblock having the largest priority number.

The remaining exterior macroblocks (not located next to any boundary macroblocks) are filled with $2^{bits_per_pixel-1}$. For 8-bit luminance component and associated chrominance this implies filling with 128.

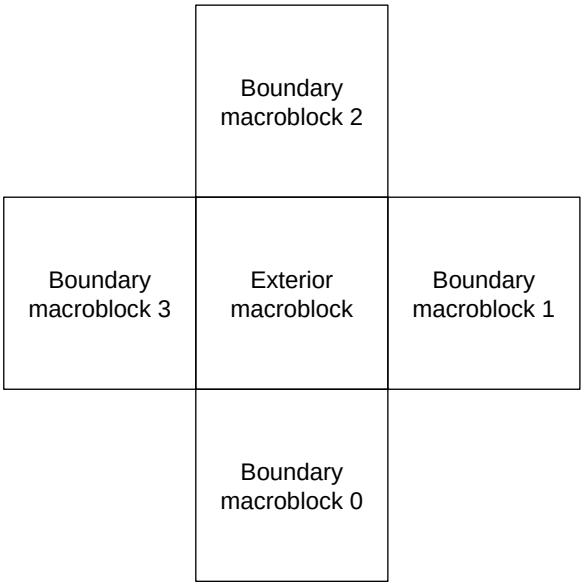


Figure 7-28 — Priority of boundary macroblocks surrounding an exterior macroblock

7.6.1.4 Padding for chrominance components

Chrominance components are padded according to subclauses 7.6.1.1 through 7.6.1.3 for each 8 x 8 block. The padding is performed by referring to a shape block generated by decimating the shape block of the corresponding luminance component. This decimating of the shape block is performed by the subsampling process described in subclause 6.1.3.6.

7.6.1.5 Padding of interlaced macroblocks

Macroblocks of interlaced VOP (interlaced = 1) are padded according to subclauses 7.6.1.1 through 7.6.1.3. The vertical padding of the luminance component, however, is performed for each field independently. A sample outside of a VOP is therefore filled with the value of the nearest boundary sample of the same field. Completely transparent blocks are padded with $2^{bits_per_pixel-1}$. Chrominance components of interlaced VOP are padded according to subclause 7.6.1.4, however, based on fields to enhance subjective quality of display in 4:2:0 format. The padding method described in this subclause is not used outside the bounding rectangle of the VOP.

7.6.1.6 Vector padding technique

The vector padding technique is applied to generate the vectors for the transparent blocks within a non-transparent macroblock, for an INTRA-coded macroblock and for a skipped macroblock included in a P-VOP. It works in a similar way as the horizontal followed by the vertical repetitive padding, and can be regarded as the repetitive padding performed on a 2x2 block except that the padded values are two dimensional vectors. A macroblock has

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

four 8x8 luminance blocks, let {MVx[i], MVy[i], i=0,1,2,3} and {Transp[i], i=0,1,2,3} be the vectors and the transparencies of the four 8x8 blocks, respectively, the vector padding is any process numerically equivalent to:

```
if (the macroblock is INTRA-coded, or skipped and included in a P-VOP) {
    MVx[0] = MVx[1] = MVx[2] = MVx[3] = 0
    MVy[0] = MVy[1] = MVy[2] = MVy[3] = 0
} else {
    if(Transp[0] == TRANSPARENT) {
        MVx[0]=(Transp[1] != TRANSPARENT) ? MVx[1] :((Transp[2]!=TRANSPARENT) ?
        MVx[2]:MVx[3]));
        MVy[0]=(Transp[1] != TRANSPARENT) ? MVy[1]:((Transp[2]!=TRANSPARENT) ?
        MVy[2]:MVy[3]));
    }
if(Transp[1] == TRANSPARENT) {
    MVx[1]=(Transp[0] != TRANSPARENT) ? MVx[0] :((Transp[3]!=TRANSPARENT) ?    MVx[3]:MVx[2]));
    MVy[1]=(Transp[0] != TRANSPARENT) ? MVy[0]:((Transp[3]!=TRANSPARENT) ?    MVy[3]:MVy[2]));
}
if(Transp[2] == TRANSPARENT) {
    MVx[2]=(Transp[3] != TRANSPARENT) ? MVx[3] :((Transp[0]!=TRANSPARENT) ?    MVx[0]:MVx[1]));
    MVy[2]=(Transp[3] != TRANSPARENT) ? MVy[3]:((Transp[0]!=TRANSPARENT) ?    MVy[0]:MVy[1]));
}
if(Transp[3] == TRANSPARENT) {
    MVx[3]=(Transp[2] != TRANSPARENT) ? MVx[2] :((Transp[1]!=TRANSPARENT) ?    MVx[1]:MVx[0]));
    MVy[3]=(Transp[2] !=TRANSPARENT) ? MVy[2]:((Transp[1]!=TRANSPARENT) ? MVy[1]:MVy[0]));
    }
}
```

Vector padding is only used in I-, P-, and S(GMC)-VOPs, it is applied on a macroblock directly after it is decoded. The block vectors after padding are used in the P-, and S(GMC)-VOP vector decoding and binary shape decoding, and in the B-VOP direct mode decoding. Note that the averaged motion vector described in 7.8.7.3 is used as the motion vectors of a GMC macroblock (i.e. a macroblock included in an S (GMC)-VOP and mcsel == '1') for this padding process.

7.6.2 Sample interpolation for non-integer motion vectors

The sample interpolation in case of non-integer motion vectors is performed for 16x16 or 8x8 frame blocks in case of progressive and for 16x8 field blocks in case of interlaced macroblocks. It is done using the luminance and chrominance components of the respectively reconstructed and padded reference VOP.

The value for the quarter_sample flag is defined in the VOL header (see subclause 6.2.4), the value of rounding_control is defined using the vop_rounding_type bit in the vop header (see subclause 6.3.5). Note that the samples outside the padded region cannot be used for interpolation. For interlaced video, the half and quarter sample values are vertically interpolated between two successive lines of the same field. The field motion vectors are given in frame coordinates; that is the vertical coordinates of the integer samples differ by 2.

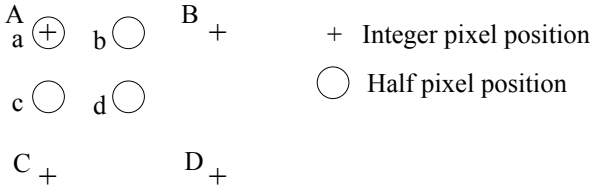
7.6.2.1 Half sample mode interpolation

The process for interpolation of half sample values defined in this subclause is carried out only in half sample mode, i.e. when quarter_sample==0. Here, the half sample values are calculated by bilinear interpolation as depicted in Figure 7-29. The value of rounding_control is defined using the vop_rounding_type bit in the VOP header (see subclause 6.3.5). Note that the samples outside the padded region cannot be used for interpolation.

--`,,`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



a = A,
b = (A + B + 1 - rounding_control) / 2
c = (A + C + 1 - rounding_control) / 2,
d = (A + B + C + D + 2 - rounding_control) / 4

Figure 7-29 — Interpolation scheme for half sample search

7.6.2.2 Quarter sample mode interpolation

The process for interpolation of half and quarter sample values defined in this subclause is carried out only in quarter sample mode, i.e. when quarter_sample==1. Further, the process is only carried out for the luminance component of a VOP; for motion compensation of the chrominance components, the respective motion vectors are rounded to the next half pel position (see Table 7-13) and then processed as described in half pel sample interpolation (subclause 7.6.2.1).

In quarter sample mode interpolation, for each block of size MxN in the reference VOP which position is defined by the decoded motion vector for the block to be predicted, a reference block of size (M+1)x(N+1) biased in the direction of the half or quarter sample position is read from the reconstructed and padded reference VOP. Then, this reference block is symmetrically extended at the block boundaries by three samples using block boundary mirroring according to Figure 7-30.

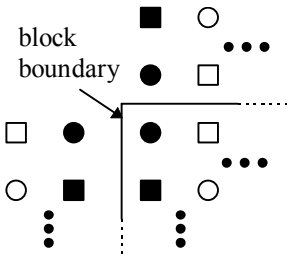


Figure 7-30 — block boundary mirroring

The half sample values and (if applicable) the quarter sample values are then calculated from this extended reference block in two steps described in the following.

7.6.2.2.1 Calculation of the half sample values

Figure 7-31 shows integer and half sample positions. The half sample values are calculated by horizontal filtering to obtain values at positions b_i followed by vertical filtering to obtain the values at positions c and d. For the filtering in both directions an 8-tap FIR interpolation filter described by

(C[4], C[3], C[2], C[1], C[1], C[2], C[3], C[4])/256

where C[1..4]= [160, -48, 24, -8] is used. The equations for interpolation of half sample positions are

$$a = A_{-1,-1},$$

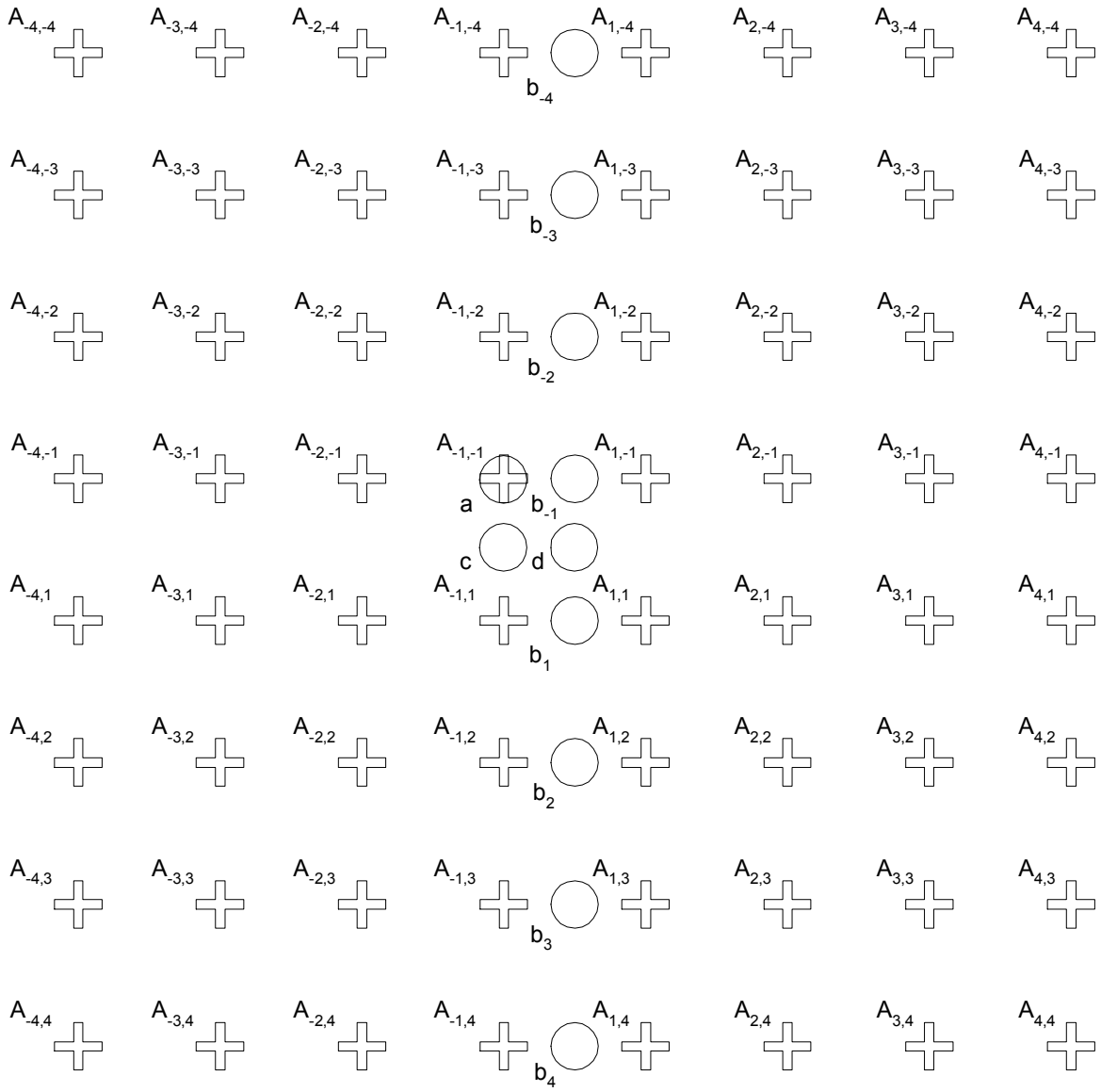
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$b_i = \left(\left(\sum_{j=1}^{4} C[j] \cdot (A_{-j,i} + A_{+j,i}) \right) + 128 - rounding_control \right) / 256,$$

$$c = \left(\left(\sum_{j=1}^{4} C[j] \cdot (A_{-1,-j} + A_{-1,+j}) \right) + 128 - rounding_control \right) / 256,$$

and

$$d = \left(\left(\sum_{j=1}^{4} C[j] \cdot (b_{-j} + b_{+j}) \right) + 128 - rounding_control \right) / 256.$$



+: Integer sample position; o: Half sample position

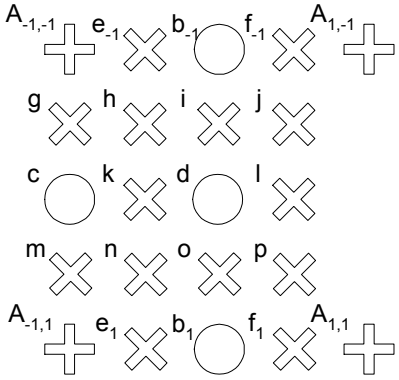
Figure 7-31 — Half sample interpolation in quarter sample mode (FIR filter)

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The values generated by both horizontal and vertical filtering pass are clipped to the range of [0, $2^{bits_per_pixel}$-1]. The value of rounding_control is defined using the vop_rounding_type bit in the vop header (see subclause 6.3.5).

7.6.2.2.2 Calculation of the quarter sample values

If applicable, the quarter sample values are calculated in a second step following the half sample interpolation described above. Figure 7-32 shows integer, half, and quarter sample positions.



+: Integer sample position; o: Half sample position; x: Quarter sample position

Figure 7-32 — Quarter sample interpolation in quarter sample mode (FIR and bilinear filter)

The quarter sample values are approximately bilinear interpolations of the half sample positions. However, because of the nonlinear operations of rounding and clipping, the interpolation is not perfectly bilinear. The interpolation shall be accomplished by any method equivalent to the following equations:

$$e_i = \left(A_{-1,i} + b_i + 1 - rounding_control\right)/2$$

$$f_i = \left(b_i + A_{1,i} + 1 - rounding_control\right)/2$$

$$k = \left(\sum_{j=1}^{4} C[j] \cdot (e_{-j} + e_{+j}) + 128 - rounding_control\right)/256$$

$$l = \left(\sum_{j=1}^{4} C[j] \cdot (f_{-j} + f_{+j}) + 128 - rounding_control\right)/256$$

$$g = \left(A_{-1,-1} + c + 1 - rounding_control\right)/2$$

$$h = \left(e_{-1} + k + 1 - rounding_control\right)/2$$

$$i = \left(b_{-1} + d + 1 - rounding_control\right)/2$$

$$j = \left(l + f_{-1} + 1 - rounding_control\right)/2$$

$$m = \left(c + A_{-1,1} + 1 - rounding_control\right)/2$$

$$n = \left(k + e_1 + 1 - rounding_control\right)/2$$

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$o = \left(d + b_1 + 1 - rounding_control\right)/2$$

$$p = \left(l + f_1 + 1 - rounding_control\right)/2$$

The values computed are clipped to the range of [0, $2^{bits_per_pixel}$-1]. The value of rounding_control affects the direction of rounding.

7.6.3 General motion vector decoding process

To decode a motion vector (MVx, MVy), the differential motion vector (MVDx, MVDy) shall be extracted from the bitstream by using the variable length decoding as described in subclause 6.3.6.2. Then it shall be added component-wise to a motion vector predictor (Px, Py) to form the final motion vector. The general motion vector decoding process shall be any process that is equivalent to the following one:

All calculations are carried out as unitless integers. If quarter_sample==0, then the integer values represent halfsample units; if quarter_sample==1, then the integer values represent quartersample units. This process is valid for the motion vector decoding in interlaced/progressive P-, S(GMC)- and B-VOPs except that the generation of the predictor (Px, Py) may be different.

```
r_size = vop_fcode - 1
f = 1 << r_size
high = ( 32 * f ) -  1;
low = ( (-32) * f );
range = ( 64 * f );

if ( (f  == 1) || (horizontal_mv_data == 0) )
     MVDx = horizontal_mv_data;
else {
     MVDx = ( ( Abs(horizontal_mv_data) - 1 ) * f ) + horizontal_mv_residual + 1;
     if (horizontal_mv_data < 0)
          MVDx = -MVDx;
}

if ( (f  == 1) || (vertical_mv_data == 0) )
     MVDy = vertical_mv_data;
else {
     MVDy = ( ( Abs(vertical_mv_data) - 1 ) * f ) + vertical_mv_residual + 1;
     if (vertical_mv_data < 0)
          MVDy = -MVDy;
}

MVx = Px + MVDx;
if ( MVx < low )
     MVx = MVx + range;
if (MVx > high)
     MVx = MVx - range;

MVy = Py + MVDy;
if ( MVy < low )
     MVy = MVy + range;
if (MVy > high)
     MVy = MVy - range;
```

The value of mv_data (i.e., horizontal_mv_data and vertical_mv_data) is equal to two times the value found in the 'vector differences' column of Table B.12 associated with the received codeword.

The parameters in the bitstream shall be such that the components of the reconstructed differential motion vector, MVDx and MVDy, shall lie in the range [low:high], at the time of their use in calculating the values of MVx and MVy

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(i.e., intermediate values of MVDx and MVDy may occur that are outside the range [low:high]). In addition the components of the reconstructed motion vector, *MVx* and *MVy,* shall also lie in the range [*low* : *high*]. The allowed range [low : high] for the motion vectors depends on the parameter vop_fcode; it is shown in Table 7-9.

The variables *r_size*, *f*, *MVDx, MVDy*, *high*, *low* and *range* are temporary variables that are not used in the remainder of this part of ISO/IEC 14496. The parameters horizontal_mv_data, vertical_mv_data, horizontal_mv_residual and vertical_mv_residual are parameters recovered from the bitstream.

The variable *vop_fcode* refers either to the parameter vop_fcode_forward or to the parameter vop_fcode_backward which have been recovered from the bitstream, depending on the respective prediction mode. In the case of P- or S(GMC)-VOP prediction only forward prediction applies. In the case of B-VOP prediction, forward as well as backward prediction may apply.

Table 7-9 — Range for motion vectors

vop_fcode_forward or vop_fcode_backward	motion vector range in unitless integers [low:high]
1	[-32,31]
2	[-64,63]
3	[-128,127]
4	[-256,255]
5	[-512,511]
6	[-1024,1023]
7	[-2048,2047]

If the current macroblock is a field motion compensated macroblock, then the same prediction motion vector (Px, Py) is used for both field motion vectors. Because the vertical component of a field motion vector is an even integer, the vertical differential motion vector encoded in the bitstream is

$$MVy = MVDy_{field} + PY / 2.$$

7.6.4 Unrestricted motion compensation

Motion vectors are allowed to point outside the decoded area of a reference VOP when (and only when) the short video header format is not in use (i.e., when short_video_header is 0). For an arbitrary shape VOP, the decoded area refers to the area within the bounding rectangle, padded as described in subclause 7.6.1. A bounding rectangle is defined by vop_width and vop_height extended to multiple of 16. In case of half sample mode, when a sample referenced by a motion vector stays outside the decoded VOP area, an edge sample is used. This edge sample is retrieved by limiting the motion vector to the last full pel position inside the decoded VOP area. Limitation of a motion vector is performed on a sample basis and separately for each component of the motion vector, as depicted in Figure 7-33. In case of quarter sample mode, when a sample needed for interpolation (see subclause 7.6.2.2) stays outside the decoded VOP area, an edge sample is used prior to block boundary mirroring.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

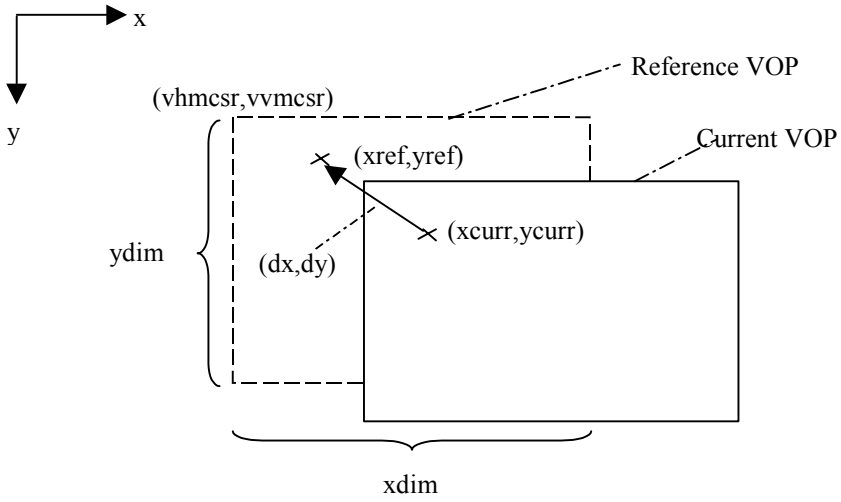


Figure 7-33 — Unrestricted motion compensation

The coordinates of a reference sample in the reference VOP, (yref, xref) is determined as follows :

$$xref = MIN (MAX (xcurr+dx, vhmcsr), xdim+vhmcsr-1)$$

$$yref = MIN (MAX (ycurr+dy, vvmcsr), ydim+vvmcsr-1)$$

where vhmcsr = vop_horizontal_mc_spatial_ref, vvmcsr = vop_vertical_mc_spatial_ref, (ycurr, xcurr) are the coordinates of a sample in the current VOP, (yref, xref) are the coordinates of a sample in the reference VOP, (dy, dx) is the motion vector, and (ydim, xdim) are the dimensions of the bounding rectangle of the reference VOP. All coordinates are related to the absolute coordinate system shown in Figure 7-33. Note that for rectangular VOP, a reference VOP is defined by video_object_layer_width and video_object_layer_height, extended to a multiple of 16. For an arbitrary shape VOP, a reference VOP of luminance is defined by vop_width and vop_height extended to a multiple of 16, while that of chrominance is defined by (vop_width>>1) and (vop_height>>1) extended to a multiple of 8.

7.6.5 Vector decoding processing and motion-compensation in progressive P- and S(GMC)-VOP

An inter-coded macroblock comprises either one motion vector for the complete macroblock or K (1< K<=4) motion vectors, one for each non-transparent 8x8 pel blocks forming the 16x16 pel macroblock, as is indicated by the mcbpc code.

For decoding a motion vector, the horizontal and vertical motion vector components are decoded differentially by using a prediction, which is formed by a median filtering of three vector candidate predictors (MV1, MV2, MV3) from the spatial neighbourhood macroblocks or blocks already decoded. Note that if any spatial neighbourhood macroblock is a GMC macroblock (i.e. included in an S (GMC)-VOP and mcsel == '1') and it is in the current VOP and video packet, the averaged motion vector described in subclause 7.8.7.3 is used as candidate motion vectors. The spatial position of candidate predictors for each block vector is depicted in Figure 7-34. In the case of only one motion vector present for the complete macroblock, the top-left case in Figure 7-34 is applied. When the short video header format is in use (i.e., when short_video_header is "1"), only one motion vector shall be present for a macroblock.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

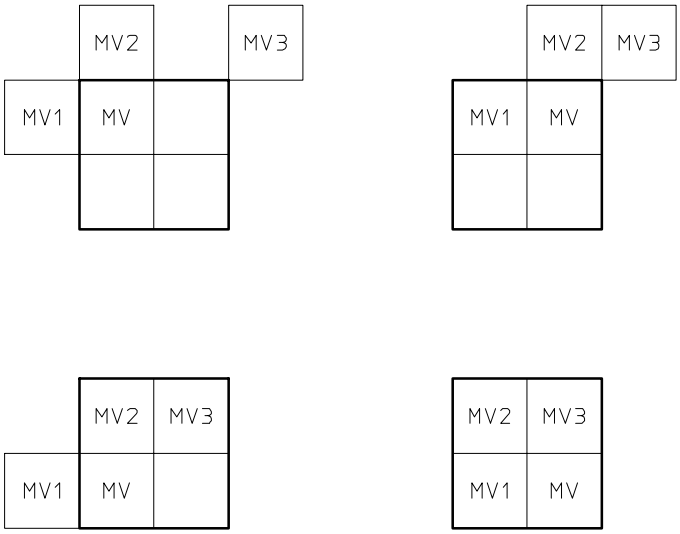


Figure 7-34 — Definition of the candidate predictors MV1, MV2 and MV3 for each of the luminance blocks in a macroblock

The following four decision rules are applied to obtain the value of the three candidate predictors:

- If a candidate predictor MVi is in a transparent spatial neighbourhood macroblock or in a transparent block of the current macroblock it is not valid, otherwise, it is set to the corresponding block vector.
- If one and only one candidate predictor is not valid, it is set to zero.
- If two and only two candidate predictors are not valid, they are set to the third candidate predictor.
- If all three candidate predictors are not valid, they are set to zero.

Note that any neighbourhood macroblock outside the current VOP or video packet or outside the current GOB (when short_video_header is “1”) for which gob_header_empty is “0” is treated as transparent in the above sense. The median value of the three candidates for the same component is computed as predictor, denoted by Px and Py:

$$Px = Median(MV1x, MV2x, MV3x)$$
$$Py = Median(MV1y, MV2y, MV3y)$$

For instance, if MV1=(-2,3), MV2=(1,5) and MV3=(-1,7), then Px = -1 and Py = 5. The final motion vector is then obtained by using the general decoding process defined in the subclause 7.6.3.

If four vectors are used, each of the motion vectors is used for all pixels in one of the four luminance blocks in the macroblock. The numbering of the motion vectors is equivalent to the numbering of the four luminance blocks as given in Figure 6-8. Motion vector MVD_{CHR} for both chrominance blocks is derived by calculating the sum of the *K* luminance vectors, that corresponds to *K* 8x8 blocks that do not lie outside the VOP shape and dividing this sum by 2**K;* in quarter sample mode the vectors are divided by 2 before summation. The component values of the resulting sixteenth/twelfth/eighth/fourth sample resolution vectors are modified towards the nearest half sample position as indicated below.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-10 — Modification of sixteenth sample resolution chrominance vector components

sixteenth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	//16
resulting position	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	//2

Table 7-11 — Modification of twelfth sample resolution chrominance vector components

twelfth pixel position	0	1	2	3	4	5	6	7	8	9	10	11	//12
resulting position	0	0	0	1	1	1	1	1	1	1	2	2	//2

Table 7-12 — Modification of eighth sample resolution chrominance vector components

eighth pixel position	0	1	2	3	4	5	6	7	//8
resulting position	0	0	1	1	1	1	1	2	//2

Table 7-13 — Modification of fourth sample resolution chrominance vector components

fourth pixel position	0	1	2	3	//4
resulting position	0	1	1	1	//2

Half sample values are found using bilinear interpolation as described in subclause 7.6.2. The prediction for luminance is obtained by overlapped motion compensation as described in subclause 7.6.6 if indicated by obmc_disable==0. The prediction for chrominance is obtained by applying the motion vector MVD_{CHR} to all pixels in the two chrominance blocks.

7.6.6 Overlapped motion compensation

This subclause specifies the overlapped motion compensation process. This process is performed when the flag obmc_disable=0. Note that overlapped motion compensation is disabled over the boundary between macroblocks with different values for mcsel in an S (GMC)-VOP.

Each pixel in an 8*8 luminance prediction block is a weighted sum of three prediction values, divided by 8 (with rounding). In order to obtain the three prediction values, three motion vectors are used: the motion vector of the current luminance block, and two out of four "remote" vectors:

- the motion vector of the block at the left or right side of the current luminance block;
- the motion vector of the block above or below the current luminance block.

For each pixel, the remote motion vectors of the blocks at the two nearest block borders are used. This means that for the upper half of the block the motion vector corresponding to the block above the current block is used, while for the lower half of the block the motion vector corresponding to the block below the current block is used. Similarly, for the left half of the block the motion vector corresponding to the block at the left side of the current block is used, while for the right half of the block the motion vector corresponding to the block at the right side of the current block is used.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The creation of each pixel, $\bar{p}(i,j)$, in an 8*8 luminance prediction block is governed by the following equation:

$$\bar{p}(i,j) = (q(i,j) \times H_0(i,j) + r(i,j) \times H_1(i,j) + s(i,j) \times H_2(i,j) + 4) / /8,$$

where $q(i,j)$, $r(i,j)$, and $s(i,j)$ are the pixels from the referenced picture as defined by

$$q(i,j) = p(i + MV_x^0, j + MV_y^0),$$
$$r(i,j) = p(i + MV_x^1, j + MV_y^1),$$
$$s(i,j) = p(i + MV_x^2, j + MV_y^2).$$

Here, (MV_x^0, MV_y^0) denotes the motion vector for the current block, (MV_x^1, MV_y^1) denotes the motion vector of the block either above or below, and (MV_x^2, MV_y^2) denotes the motion vector either to the left or right of the current block as defined above.

The matrices $H_0(i,j), H_1(i,j)$ and $H_2(i,j)$ are defined in Figure 7-35, Figure 7-36, and Figure 7-37, where (i,j) denotes the column and row, respectively, of the matrix.

If one of the surrounding blocks was not coded, the corresponding remote motion vector is set to zero. If one of the surrounding blocks was coded in intra mode, the corresponding remote motion vector is replaced by the motion vector for the current block. If the current block is at the border of the VOP and therefore a surrounding block is not present, the corresponding remote motion vector is replaced by the current motion vector. In addition, if the current block is at the bottom of the macroblock, the remote motion vector corresponding with an 8*8 luminance block in the macroblock below the current macroblock is replaced by the motion vector for the current block.

4	5	5	5	5	5	5	4
5	5	5	5	5	5	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	6	6	6	6	5	5
5	5	5	5	5	5	5	5
4	5	5	5	5	5	5	4

Figure 7-35 — Weighting values, H_0, for prediction with motion vector of current luminance block

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2	2	2	2	2	2	2	2
1	1	2	2	2	2	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	2	2	2	2	1	1
2	2	2	2	2	2	2	2

Figure 7-36 — Weighting values, H1, for prediction with motion vectors of the luminance blocks on top or bottom of current luminance block

2	1	1	1	1	1	1	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	2	1	1	1	1	2	2
2	1	1	1	1	1	1	2

Figure 7-37 — Weighting values, H_2, for prediction with motion vectors of the luminance blocks to the left or right of current luminance block

7.6.7 Temporal prediction structure

A forward reference VOP is defined as a most recently decoded I- or P- or S(GMC)-VOP in the past for which "vop_coded==1". A backward reference VOP is defined as the most recently decoded I- or P- or S(GMC)-VOP in the future, regardless of its value for "vop_coded".

A target P- or S(GMC)-VOP shall make reference to the forward reference VOP

A target B-VOP can make reference

- to the forward and/or the backward reference VOP, if for the backward reference VOP "vop_coded==1"
- only to the forward reference VOP, if for the backward reference VOP "vop_coded==0"

Note that for the reference VOP selection of binary shape coding the rules stated in subclause 7.5.2.4 shall be applied

The temporal prediction structure is depicted in Figure 7-38.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

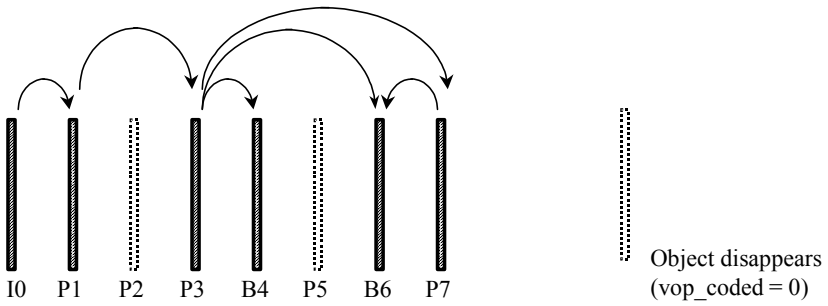


Figure 7-38 — Temporal Prediction Structure

7.6.8 Vector decoding process of non-scalable progressive B-VOPs

In B-VOPs there are three kinds of vectors, namely, 16x16 forward vector, 16x16 backward vector and the delta vector for the direct mode. The vectors are decoded with respect to the corresponding vector predictors. The basic decoding process of a differential vector is the exactly same as defined in P-VOPs except that for the delta vector of the direct mode the f_code is always one. The vector is then reconstructed by adding the decoded differential vector to the corresponding vector predictor. The vector predictor for the delta vector is always set to zero, while the forward and backward vectors have their own vector predictors, which are reset to zero only at the beginning of each macroblock row. The vector predictors are updated in the following three cases:

- after decoding a macroblock of forward mode only the forward predictor is set to the decoded forward vector
- after decoding a macroblock of backward mode only the backward predictor is set to the decoded backward vector
- after decoding a macroblock of bi-directional mode both the forward and backward predictors are updated separately with the decoded vectors of the same type (forward/backward).

7.6.9 Motion compensation in non-scalable progressive B-VOPs

In B-VOPs the overlapped motion compensation (OBMC) is not employed. The motion-compensated prediction of B-macroblock is generated by using the decoded vectors and taking reference to the padded forward/backward reference VOPs as defined below. Arbitrarily shaped reference VOPs shall be padded accordingly.

7.6.9.1 Basic motion compensation procedure

All of the ISO/IEC 14496-2 motion compensation techniques are based on the formation of a prediction block, pred[i][j] of dimension (width, height), from a reference image, ref[x][y]. The coordinates of the current block (or macroblock) in the reference VOP is (x,y), the motion half-pel resolution motion vector is (dx_halfpel, dy_halfpel). The pseudo-code for this procedure is given below. For quarter pel mode the formation of the prediction block is carried out as described in subclause 7.6.2.2.

The component_width() and component_height() function give the coded VOP dimensions for the current component. For luminance, component_width() is video_object_layer_width for a rectangular VOP or vop_width otherwise rounded up to the next multiple of 16. The luminance component_height() is defined similarly. The chrominance dimensions are one half of the corresponding luminance dimension.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
clip_ref(ref, x, y)
{
    return(ref[MIN(MAX(x, 0), component_width(ref) - 1)]
              [MIN(MAX(y, 0), component_height(ref) - 1)]);
}
mc(pred,                        /* prediction block */
   ref,                         /* reference component */
   x, y,                        /* ref block coords for MV=(0, 0) */
   width, height,               /* reference block dimensions */
   dx_halfpel, dy_halfpel,      /* half-pel resolution motion vector */
   rounding,                    /* rounding control (0 or 1 ) */
   pred_y0,                     /* field offset in pred blk (0 or 1) */
   ref_y0,                      /* field offset in ref blk (0 or 1) */
   y_incr)                      /* vertical increment (1 or 2) */
{
    dx = dx_halfpel >> 1;
    dy = y_incr * (dy_halfpel >> y_incr);
    if (dy_halfpel & y_incr) {
        if (dx_halfpel & 1) {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                        (clip_ref(ref, x_ref + 0, y_ref + 0) +
                         clip_ref(ref, x_ref + 1, y_ref + 0) +
                         clip_ref(ref, x_ref + 0, y_ref + y_incr) +
                         clip_ref(ref, x_ref + 1, y_ref + y_incr) +
                         2 - rounding) >> 2;
                }
            }
        } else {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                        (clip_ref(ref, x_ref, y_ref + 0) +
                         clip_ref(ref, x_ref, y_ref + y_incr) +
                         1 - rounding) >> 1;
                }
            }
        }
    } else {
        if (dx_halfpel & 1) {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                        (clip_ref(ref, x_ref + 0, y_ref) +
                         clip_ref(ref, x_ref + 1, y_ref) +
                         1 - rounding) >> 1;
                }
            }
        } else {
            for (iy = 0; iy < height; iy += y_incr) {
                for (ix = 0; ix < width; ix++) {
                    x_ref = x + dx + ix;
                    y_ref = y + dy + iy + ref_y0;
                    pred[ix][iy + pred_y0] =
                        clip_ref(ref, x_ref, y_ref);
                }
            }
        }
    }
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.9.2 Forward mode

Only the forward vector (MVFx,MVFy) is applied in this mode. The prediction blocks Pf_Y, Pf_U, and Pf_V are generated from the forward reference VOP, ref_Y_for for luminance component and ref_U_for and ref_V_for for chrominance components, as follows:

```
mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);

mc(Pf_U, ref_U_for, x/2, y/2, 8, 8,  MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pf_V, ref_V_for, x/2, y/2, 8, 8,  MVFx_chro, MVFy_chro, 0, 0, 0,1);
```

where (MVFx_chro, MVFy_chro) is motion vector derived from the luminance motion vector by dividing each component by 2 then rounding on a basis of Table 7-13. Here (and hereafter) the function MC is defined in subclause 7.6.9.

7.6.9.3 Backward mode

Only the backward vector (MVBx,MVBy) is applied in this mode. The prediction blocks Pb_Y, Pb_U, and Pb_V are generated from the backward reference VOP, ref_Y_back for luminance component and ref_U_back and ref_V_back for chrominance components, as follows:

```
mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);

mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);
```

where (MVBx_chro, MVBy_chro) is motion vector derived from the luminance motion vector by dividing each component by 2 then rounding on a basis of Table 7-13.

7.6.9.4 Bi-directional mode

Both the forward vector (MVFx,MVFy) and the backward vector (MVBx,MVBy) are applied in this mode. The prediction blocks Pi_Y, Pi_U, and Pi_V are generated from the forward and backward reference VOPs by doing the forward prediction, the backward prediction and then averaging both predictions pixel by pixel as follows.

```
mc(Pf_Y, ref_Y_for, x, y, 16, 16, MVFx, MVFy, 0, 0, 0, 1);

mc(Pf_U, ref_U_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pf_V, ref_V_for, x/2, y/2, 8, 8, MVFx_chro, MVFy_chro, 0, 0, 0,1);

mc(Pb_Y, ref_Y_back, x, y, 16, 16, MVBx, MVBy, 0, 0, 0, 1);

mc(Pb_U, ref_U_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

mc(Pb_V, ref_V_back, x/2, y/2, 8, 8, MVBx_chro, MVBy_chro, 0, 0, 0,1);

Pi_Y[i][j] = (Pf_Y[i][j] + Pb_Y[i][j] + 1)>>1;          i,j=0,1,2…15;

Pi_U[i][j] = (Pf_U[i][j] + Pb_U[i][j] + 1)>>1;      i,j=0,1,2…7;

Pi_V[i][j] = (Pf_V[i][j] + Pb_V[i][j] + 1)>>1;          i,j=0,1,2…7;
```

where (MVFx_chro, MVFy_chro) and (MVBx_chro, MVBy_chro) are motion vectors derived from the forward and backward luminance motion vectors by dividing each component by 2 then rounding on a basis of Table 7-13, respectively.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.9.5 Direct mode

This mode uses direct bi-directional motion compensation derived by employing I-, P-, or S(GMC)-VOP macroblock motion vectors and scaling them to derive forward and backward motion vectors for macroblocks in B-VOP. This is the only mode which makes it possible to use motion vectors on 8x8 blocks in B-VOPs. Only one delta motion vector is allowed per macroblock.

7.6.9.5.1 Formation of motion vectors for the direct mode

The direct mode utilises the motion vectors (MVs) of the co-located macroblock in the most recently I-, P-, or S(GMC)- VOP. The co-located macroblock is defined as the macroblock which has the same horizontal and vertical index with the current macroblock in the B-VOP. The averaged motion vector defined in subclause 7.8.7.3 is used as the MV of the co-located macroblock when this co-located macroblock is included in an S(GMC)-VOP and mcsel == '1' (i.e. when the co-located macroblock is a GMC macroblock). The MV vectors are the block vectors of the co-located macroblock after applying the vector padding defined in subclause 7.6.1.6. If the co-located macroblock is transparent and thus the MVs are not available, the direct mode is still enabled by setting MV vectors to zero vectors.

7.6.9.5.2 Calculation of vectors

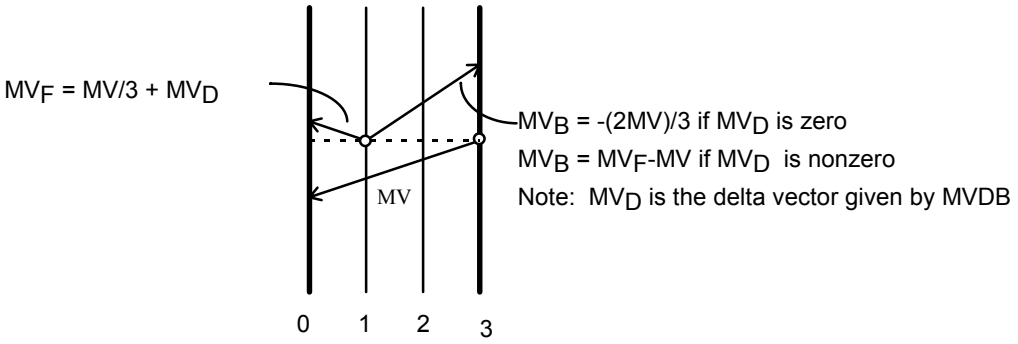


Figure 7-39 — Direct Bi-directional Prediction

Figure 7-39 shows scaling of motion vectors. The calculation of forward and backward motion vectors involves linear scaling of the collocated block in temporally next I-, P-, or S(GMC)-VOP, followed by correction by a delta vector (MVDx,MVDy). The forward and the backward motion vectors are {(MVFx[i],MVFy[i]), (MVBx[i],MVBy[i]), i = 0,1,2,3} and are given in half or quarter sample units as follows.

MVFx[i] = (TRB x MVx[i]) / TRD + MVDx

MVBx[i] = (MVDx==0)? ((TRB - TRD) x MVx[i]) / TRD : MVFx[i] - MVx[i]

MVFy[i] = (TRB x MVy[i]) / TRD + MVDy

MVBy[i] = (MVDy==0)? ((TRB - TRD) x MVy[i]) / TRD : MVFy[i] - MVy[i]

i = 0,1,2,3.

where {(MVx[i],MVy[i]), i = 0,1,2,3} are the MV vectors of the co-located macroblock, TRD is the difference in temporal reference of the B-VOP and the previous reference VOP. TRD is the difference in temporal reference of the temporally next reference VOP with temporally previous reference VOP, assuming B-VOPs or skipped VOPs in between. In case that the MV components of the co-located macroblock are given in quarter sample units and the components MVDx and MVDy of the delta vector are given in half sample units, the components of the co-located macroblock {(MVx[i],MVy[i]), i = 0,1,2,3} are converted to half sample units before the calculation of the forward and the backward motion vectors {(MVFx[i], MVFy[i]), (MVBx[i], MVBy[i]), i = 0,1,2,3}. For this conversion, each component {(MVx[i],MVy[i]), i = 0,1,2,3} is first divided by 2 and then rounded on the basis of Table 7-13.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.9.5.3 Generation of prediction blocks

Motion compensation for luminance is performed individually on 8x8 blocks to generate a macroblock. The process of generating a prediction block consists of using computed forward and backward motion vectors {(MVFx[i],MVFy[i]), (MVBx[i],MVBy[i]), i = 0,1,2,3} to obtain appropriate blocks from reference VOPs and averaging these blocks, same as the case of bi-directional mode except that motion compensation is performed on 8x8 blocks. Note that in the case of quarter sample mode, the motion compensation of four 8x8 blocks, using the same motion vector for each block, does not lead to the same result as the motion compensation of a single 16x16 block using the same vector. This is due to the block boundary mirroring described in subclause 7.6.2.2.

For the motion compensation of both chrominance blocks, the forward motion vector (MVFx_chro, MVFy_chro) is calculated by the sum of K forward luminance motion vectors dividing by 2K and then rounding toward the nearest half sample position as defined in Table 7-10 to Table 7-13. In quarter sample mode the vectors are divided by 2 before summation. The backward motion vector (MVBx_chro, MVBy_chro) is derived in the same way. The rest process is the same as the chrominance motion compensation of the bi-directional mode described in subclause 7.6.9.4.

7.6.9.6 Motion compensation in skipped macroblocks

If the co-located macroblock in the most recently decoded I- or P-VOP is skipped, the current B-macroblock is treated as the forward mode with the zero motion vector (MVFx,MVFy). If the modb equals to '1' the current B-macroblock is reconstructed by using the direct mode with zero delta vector. If the co-located macroblock in the most recently decoded S(GMC)-VOP is skipped, this co-located macroblock is treated as a non-skipped macroblock with the averaged motion vector defined in subclause 7.8.7.3 for the current B-macroblock.

7.6.10 Motion Compensation Decoding of Reduced Resolution VOP

This subclause describes motion compensation decoding process for VOP with vop_reduced_resolution equal to "1". In this case, motion compensation is done with Macroblock size of 32 x 32 pixels or Block size of 16 x 16 pixels instead of Macroblock size of 16 x 16 pixels or Block size of 8 x 8 pixels for the case of normal resolution VOP(vop_reduced_resolution is set to "0" or vop_redcuced_resolution flag is not present). Figure 7-40 shows block diagram of motion compensation decoding process of Reduced Resolution VOP.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

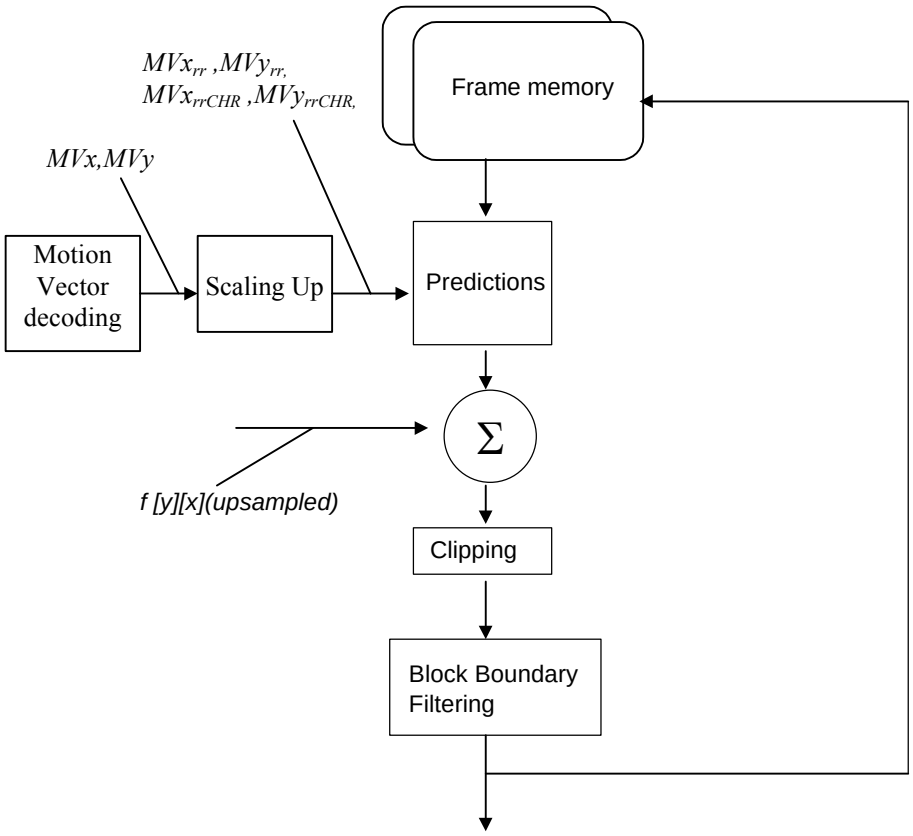


Figure 7-40 — Motion Compensation Decoding of Reduced Resolution VOP

7.6.10.1 Decoding Process of Reduced Resolution P-VOP

7.6.10.1.1 Motion Vector Decoding

Motion vector for luminance component *MVx, MVy* is decoded as in the same way as described in subclause 7.6.3. Four vector case, motion vector for each block is also decoded as described in 7.6.5. These decoded vectors for luminance are used for the candidate of the motion vector predictor in the decoding process of the following Macroblocks.

7.6.10.1.2 Scaling up of Motion Vectors

Decoded luminance motion vector *MVx, MVy* shall be scaled up to MVx_{rr}, MVy_{rr} by the following formula:

$$MVx_{rr} = 0 \qquad \text{if } MVx = 0$$
$$MVx_{rr} = \text{Sign}(MVx) * (2.0 * \text{Abs}(MVx) - 0.5) \qquad \text{if } MVx \neq 0$$

$$MVy_{rr} = 0 \qquad \text{if } MVy = 0$$
$$MVy_{rr} = \text{Sign}(MVy) * (2.0 * \text{Abs}(MVy) - 0.5) \qquad \text{if } MVy \neq 0$$

As a result, each vector component is restricted to have a half-integer or zero value, and the range of each motion vector component $MVx_{rr,}$ MVy_{rr} is enlarged to approximately twice the range of decoded motion vector component *MVx, MVy*.

Motion vector for chrominace blocks $MVx_{rrCHR,}$ MVy_{rrCHR} is derived by calculating the sum of the K luminance vectors MVx_{rr}, MVy_{rr}, that corresponds to K 16x16 blocks that do not lie outside the VOP shape and dividing this sum by 2*K; the component values of the resulting sixteenth/twelfth/eighth/fourth sample resolution vectors are modified towards the nearest half sample position as indicated in Table 7-10 to Table 7-13.

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.6.10.1.3 Prediction

By using the scaled up luminance motion vector MVx_{rr}, MVy_{rr} and derived chrominance vector MVx_{rrCHR}, MVy_{rrCHR} predicton data for each 32x32 Macroblock or 16x16 block is generated from the refrence VOP data stored in frame memory. Half sample interpolation shall be done as described in subclause 7.6.2.

If Overlapped Motion Compensation is also used for the Reduced Resolution VOP, enlarged matrices of weighting values are used to perform the overlapped motion compensation. Except that the size of each block and matrices of weighting values is 16×16, the procedure of the creation of each prediction block is identical to the description of subclause. 7.6.6.

The enlarged matrices of weighting values for the 16×16 luminance prediction are given in Figure 7-40, Figure 7-41, and Figure 7-42.

4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4
4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	6	6	6	6	6	6	6	6	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4
4	4	5	5	5	5	5	5	5	5	5	5	5	5	4	4

Figure 7-41 — Weighting values, H_0, for prediction with motion vector of current 16×16 luminance block

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Figure 7-42 — Weighting values, H_1, for prediction with motion vector of 16×16 luminance blocks on top or bottom of current 16×16 luminance block

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2
2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	2	2	1	1	1	1	1	1	1	1	2	2	2	2
2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2
2	2	1	1	1	1	1	1	1	1	1	1	1	1	2	2

Figure 7-43 — Weighting values, H_2, for prediction with motion vector of 16×16 luminance blocks to the left or right of current 16×16 luminance block

7.6.10.1.4 Addition and Clipping

The 16 x 16 reconstructed prediction error block data derived by upsampling process described in subclause 7.4.6 is added to the 16 x 16 prediction block data generated by prediction process defined in subclause 7.6.10.1.3. All the resulted 16 x 16 reconstructed block data shall be clipped to the range of [0: $2^{bits_per_pixel} - 1$].

7.6.10.1.5 Block Boundary Filtering

The filter operation described in this clause shall be performed along the edges of the 16×16 reconstructed blocks data.

The filtering is performed on the complete reconstructed VOP data before storing the data in the frame memory for future prediction. No filtering is performed across VOP edges and NEWPRED segment (defined in subclause 7.14.1) edges. Chrominance as well as luminance data is filtered.

If A and B are two pixel values on a line — horizontal or vertical — of the reconstructed VOP, and A belongs to one 16 × 16 block called block1 whereas B belongs to a neighboring 16 × 16 block called block2 which is to the right or below of block1. Figure 7-44 shows examples for the position of these pixels.

-`-`-`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

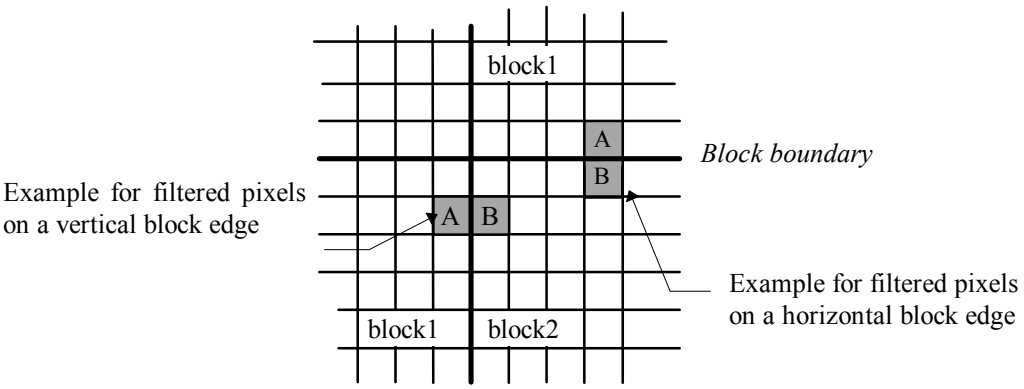


Figure 7-44 — Block boundary filter

One of the following conditions must be fulfilled in order to turn the filter on for a particular edge:

block1 belongs to a coded macroblock (not_coded == 0 || Macroblock type == INTRA) or

block2 belongs to a coded macroblock (not_coded == 0 || Macroblock type == INTRA)

A shall be replaced by A1 and B shall be replaced by B1.

A1 = (3*A + B + 2) / 4

B1 = (A + 3*B + 2) / 4

Filtering across horizontal edges: Basically this process is assumed to take place first. More precisely, the pixels $\begin{pmatrix} A \\ B \end{pmatrix}$ that are used in filtering across a horizontal edge shall not have been influenced by previous filtering across a vertical edge.

Filtering across vertical edges:

Before filtering across a vertical edge using pixels (A,B), all modifications of pixels (A,B) resulting from filtering across a horizontal edge shall have taken place.

7.6.10.2 Decoding Process of Reduced Resolution I-VOP

For I-VOP with the vop_reduced_resolution equal to “1”, Addition and Clipping process described in subclause 7.6.10.1.4 shall be performed with the fixed prediction data value of “0”. Block Boundary Filtering operation described in subclause 7.6.10.1.5 also shall be applied. The reconstructed I-VOP data is stored in the frame memory for future prediction.

7.7 Interlaced video decoding

This subclause specifies the additional decoding process that a decoder shall perform to recover VOP data from the coded bitstream when the interlaced flag in the VOL header is set to “1”. Interlaced information (subclause 6.3.6.3) specifies the method to decode bitstream of interlaced VOP.

7.7.1 Field DCT and DC and AC Prediction

When dct_type flag is set to ‘1’ (field DCT coding), DCT coefficients of luminance data are formed such that each 8x8 block consists of data from one field as being shown in Figure 6-12. DC and optional AC (see “ac_pred_flag”) prediction will be performed for a intra-coded macroblock. For the intra macroblocks which have dct_type flag being set to “1”, DC/AC prediction are performed to field blocks shown in Figure 7-45. After taking inverse DCT, all luminance blocks will be inverse permuted back to (frame) macroblock. Chrominance (block) data are not effected by dct_type flag.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

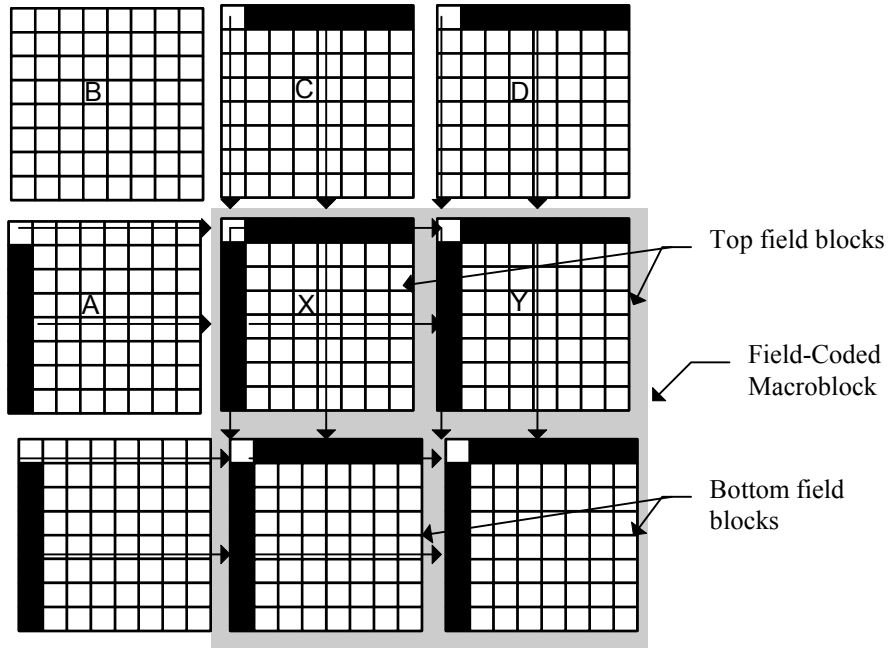


Figure 7-45 — Previous neighboring blocks used in DC/AC prediction for interlaced intra blocks.

7.7.2 Motion compensation

For non-intra macroblocks in P- and B-, and S(GMC)-VOPs, motion vectors are extracted syntactically following subclause 6.2.6 “Macroblock”. The motion vector decoding is performed separately on the horizontal and vertical components.

7.7.2.1 Motion vector decoding in P- and S(GMC)-VOP

For each component of motion vector in P- and S(GMC)-VOPs, the median value of the candidate predictor vectors for the same component is computed and add to corresponding component of the motion vector difference obtained from the bitstream. To decode the motion vectors in a P- and S(GMC)-VOP, the decoder shall first extract the differential motion vectors ($(MVDx_{f1}, MVDy_{f1})$ and $(MVDx_{f2}, MVDy_{f2})$ for top and bottom fields of a field predicted macroblock, respectively) by a use of variable length decoding and then determine the predictor vector from three candidate vectors. These candidate predictor vectors are generated from the three motion vectors of three spatial neighborhood decoded macroblocks or blocks as follows. Note that the averaged motion vector as defined in subclause 7.8.7.3 is used as the candidate predictor vector for macroblocks with mcsel == ‘1’ when the current VOP is an S(GMC)-VOP. Such macroblock is referred to as a frame predicted macroblock in the following.

CASE 1 :

If the current macroblock is a field predicted macroblock and none of the coded spatial neighborhood macroblocks is a field predicted macroblock, then candidate predictor vectors MV1, MV2, and MV3 are defined by Figure 7-46. If the candidate block *i* is not in four MV motion (8x8) mode, MV*i* represents the motion vector for the macroblock. If the candidate block *i* is in four MV motion (8x8) mode, the 8x8 block motion vector closest to the upper left block of the current MB is used. The predictors for the horizontal and vertical components are then computed by

$$P_x = Median(MV1x, MV2x, MV3x)$$
$$P_y = Median(MV1y, MV2y, MV3y).$$

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For differential motion vectors both fields use the same predictor and motion vectors are recovered by

$$MVx_{f1} = MVDx_{f1} + P_x$$
$$MVy_{f1} = 2 * (MVDy_{f1} + (P_y / 2))$$
$$MVx_{f2} = MVDx_{f2} + P_x$$
$$MVy_{f2} = 2 * (MVDy_{f2} + (P_y / 2))$$

where "/" is integer division with truncation toward 0. Note that all motion vectors described above are specified as integers with one LSB representing a half- or quarter pel displacement. The vertical component of field motion vectors always even (in half- or quarter pel frame coordinates). Vertical half- or quarter pel interpolation between adjacent lines of the same field is denoted by MVy_{fi} be an odd multiple of 2 (e.g. -2,2,6,..). No vertical interpolation is needed when MVy_{fi} is an multiple of 4 for half or 8 for quarter pel (it is a full pel value).

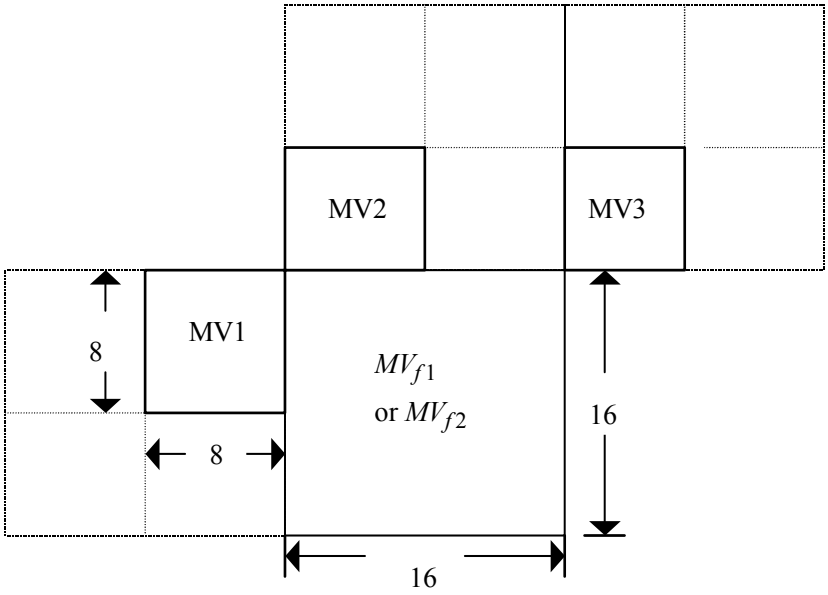


Figure 7-46 — Example of motion vector prediction for field predicted macroblocks (Case1)

CASE 2 :

If the current macroblock or block is frame predicted macroblock or block and if at least one of the coded spatial neighborhood macroblocks is a field predicted macroblock, then the candidate predictor vector for each field predicted macroblock will be generated by averaging two field motion vectors such that all fractional pel offsets are mapped into the half or quarter -pel displacement. Each component (P_x or P_y) of the final predictor vector is the median value of the candidate predictor vectors for the same component. The motion vector is recovered by

$$MVx = MVDx + P_x$$
$$MVy = MVDy + P_y.$$

where

$$P_x = Median\left(MV1x, Div2Round(MVx_{f1} + MVx_{f2}), MV3x\right),$$
$$P_y = Median\left(MV1y, Div2Round(MVy_{f1} + MVy_{f2}), MV3y\right),$$

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Div2Round(*x*) is defined as follows: *Div2Round*(*x*) = (*x* >> 1) | (*x* & 1).

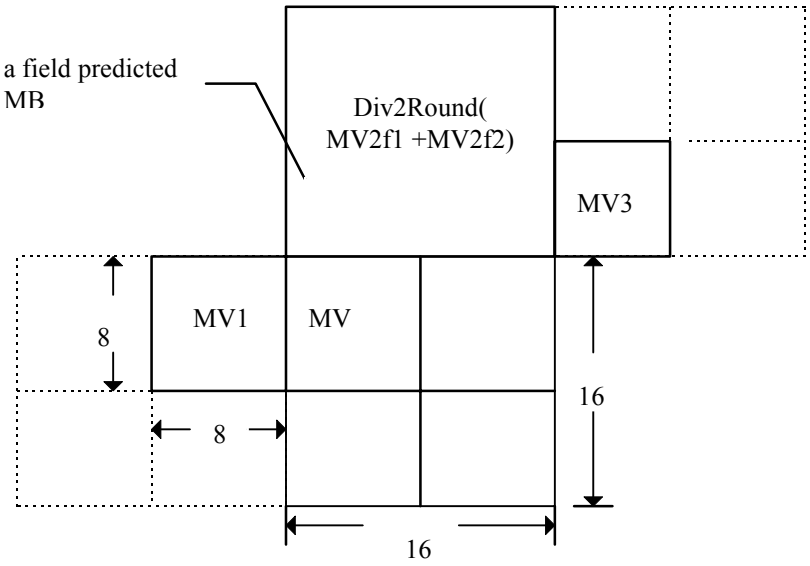


Figure 7-47 — Example of motion vector prediction for field predicted macroblocks (Case 2)

CASE 3 :

This case applies if the current macroblock is a field predicted macroblock and at least one of the coded spatial neighborhood macroblocks is a field predicted macroblock. If the candidate block *i* is field predicted, the candidate predictor vector MV*i* will be generated by averaging two field motion vectors such that all fractional pel offsets are mapped into the half- or quarter pel displacement as discribed in CASE 2. If the candidate block *i* is neither in four MV motion (8x8) mode nor in field prediction mode, MV*i* represents the frame motion vector for the macroblock. If the candidate block *i* is in four MV motion (8x8) mode, the 8x8 block motion vector closest to the upper left block of the current MB is used. The predictors for the horizontal and vertical components are then computed by

$$P_x = Median(MV1x, MV2x, MV3x)$$
$$P_y = Median(MV1y, MV2y, MV3y)$$

where

$$MVi\ x = Div2Round(MVx_{f1} + MVx_{f2}),$$
$$MVi\ y = Div2Round(MVy_{f1} + MVy_{f2}),$$

for some *i* in {1,2,3}.

For differential motion vectors both fields use the same predictor and motion vectors are recovered by (see both Figure 7-46 and Figure 7-47)

$$MVx_{f1} = MVDx_{f1} + P_x$$
$$MVy_{f1} = 2 * (MVDy_{f1} + (P_y / 2))$$
$$MVx_{f2} = MVDx_{f2} + P_x$$
$$MVy_{f2} = 2 * (MVDy_{f2} + (P_y / 2))$$

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The motion compensated prediction macroblock is calculated calling the “field_compensate_one_reference” using the motion vectors calculated above. In quarter_sample mode the macroblock is calculated as described in subclause 7.6.2.2, accordingly. The top_field_ref, bottom_field_ref, and rounding type come directly from the syntax as forward_top_field_reference, forward_bottom_field_reference and vop_rounding_type respectively. The reference VOP is defined such the even lines (0, 2, 4, ...) are the top field and the odd lines (1, 3, 5, ...) are the bottom field.

```
field_motion_compensate_one_reference(
    luma_pred, cb_pred, cr_pred, /* Prediction component pel array */
    luma_ref, cb_ref, cr_ref,    /* Reference VOP pel arrays */
    mv_top_x, mv_top_y,           /* top field motion vector */
    mv_bot_x, mv_bot_y,          /* bottom field motion vector */
    top_field_ref,               /* top field reference */
    bottom_field_ref,            /* bottom field reference */
    x, y,                        /* current luma macroblock coords */
    rounding_type)               /* rounding type */
{
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_top_x, mv_top_y,
        rounding_type, 0, top_field_ref, 2);
    mc(luma_pred, luma_ref, x, y, 16, 16, mv_bot_x, mv_bot_y,
        rounding_type, 1, bottom_field_ref, 2);
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y),
        rounding_type, 0, top_field_ref, 2);
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_top_x), Div2Round(mv_top_y),
        rounding_type, 0, top_field_ref, 2);
    mc(cb_pred, cb_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y),
        rounding_type, 1, bottom_field_ref, 2);
    mc(cr_pred, cr_ref, x/2, y/2, 8, 8,
        Div2Round(mv_bot_x), Div2Round(mv_bot_y),
        rounding_type, 1, bottom_field_ref, 2);
}
```

In the case that obmc_disable is “0”, the OBMC is not applied if the current MB is field-predicted. If the current MB is frame-predicted (including 8x8 mode) and some adjacent MBs are field-predicted, the motion vectors of those field-predicted MBs for OBMC are computed in the same manner as the candidate predictor vectors for field-predicted MBs are.

7.7.2.2 Motion vector decoding in B-VOP

For interlaced B-VOPs, a macroblock can be coded using (1) direct coding, (2) 16x16 motion compensation (includes forward, backward & bidirectional modes), or (3) field motion compensation (includes forward, backward & bidirectional modes). Motion vector in half or quarter sample accuracy will be employed for a 16x16 macroblock being coded. Chrominance vectors are derived by scaling of luminance vectors using the rounding tables described in Table 7-13 (i.e. by applying *Div2Round* to the luminance motion vectors, divided by 2 in case of quarter_sample mode). These coding modes except direct coding mode allow switching of quantiser scale from the one previously in use. Specification of dquant, a differential quantiser scale involves a 2-bit overhead. In direct coding mode, the quantiser scale value for previous coded macroblock is used.

For interlaced B-VOP motion vector predictors, four prediction motion vectors (PMVs) are used:

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-14 — Prediction motion vector allocation for interlaced P- and S(GMC)-VOPs

Function	PMV
Top field forward	0
Bottom field forward	1
Top field backward	2
Bottom field backward	3

These PMVs are used as follows for the different macroblock prediction modes:

Table 7-15 — Prediction motion vectors for interlaced B-VOP decoding

Macroblock mode	PMVs used	PMVs updated
Direct	none	none
Frame forward	0	0,1
Frame backward	2	2,3
Frame bidirectional	0,2	0,1,2,3
Field forward	0,1	0,1
Field backward	2,3	2,3
Field bidirectional	0,1,2,3	0,1,2,3

The PMVs used by a macroblock are set to the value of current macroblock motion vectors after being used.

When a frame macroblock is decoded, the two field PMVs (top and bottom field) for each prediction direction are set to the same frame value. The PMVs are reset to zero at the beginning of each row of macroblocks. The predictors are not zeroed by skipped macroblocks or direct mode macroblocks.

The frame based motion compensation modes are described in subclause 7.6. The field motion compensation modes are calculated using the “field_motion_compensate_one_reference()” pseudo code function described above. The field forward mode is denoted by mb_type == “0001” and field_prediction == “1”. The PMV update and calculation of the motion compensated prediction is shown below. The luma_fwd_ref_VOP[][], cb_fwd_ref_VOP[][], cr_fwd_ref_VOP[][] denote the entire forward (past) reference VOP pixel arrays. The coordinates of the upper left corner of the luminance macroblock is given by (x, y) and MVD[].x and MVD[].y denote an array of the motion vector differences in the order they occur in the bitstream for the current macroblock.

```
PMV[0].x = PMV[0].x + MVD[0].x;
PMV[0].y = 2 * (PMV[0].y / 2 + MVD[0].y);
PMV[1].x = PMV[1].x + MVD[1].x;
PMV[1].y = 2 * (PMV[1].y / 2 + MVD[1].y);
field_motion_compensate_one_reference(
    luma_pred, cb_pred, cr_pred,
    luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,
    PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,
    forward_top_field_reference,
    forward_bottom_field_reference,
    x, y, 0);
```

The field backward mode is denoted by mb_type == “001” and field_prediction == “1”. The PMV update and prediction calculation is outlined the following pseudo code. The luma_bak_ref_vop[][], cb_bak_ref_vop[][], cr_bak_ref_vop[][] denote the entire backward (future) reference VOP pixel arrays.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    PMV[2].x = PMV[2].x + MVD[0].x;
    PMV[2].y = 2 * (PMV[2].y / 2 + MVD[0].y);
    PMV[3].x = PMV[1].x + MVD[1].x;
    PMV[3].y = 2 * (PMV[3].y / 2 + MVD[1].y);
    field_motion_compensate_one_reference(
        luma_pred, cb_pred, cr_pred,
        luma_bak_ref_vop, cb_bak_ref_vop, cr_bak_ref_vop,
        PMV[2].x, PMV[2].y, PMV[3].x, PMV[3].y,
        backward_top_field_reference,
        backward_bottom_field_reference,
        x, y, 0);
```

The bidirectional field prediction is used when mb_type == “01” and field_prediction == “1”. The prediction macroblock (in luma_pred[][], cb_pred[][], and cr_pred[][]) is calculated by:

```
for (mv = 0; mv < 4; mv++) {
        PMV[mv].x = PMV[mv].x + MVD[mv].x;
        PMV[mv].y = 2 * (PMV[mv].y / 2 + MVD[mv].y);
    }
    field_motion_compensate_one_reference(
        luma_pred_fwd, cb_pred_fwd, cr_pred_fwd,
        luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,
        PMV[0].x, PMV[0].y, PMV[1].x, PMV[1].y,
        forward_top_field_reference,
        forward_bottom_field_reference,
        x, y, 0);
    field_motion_compensate_one_reference(
        luma_pred_bak, cb_pred_bak, cr_pred_bak,
        luma_bak_ref_vop, cb_bak_ref_vop, cr_bak_ref_vop,
        PMV[2].x, PMV[2].y, PMV[3].x, PMV[3].y,
        backward_top_field_reference,
        backward_bottom_field_reference,
        x, y, 0);
    for (iy = 0; iy < 16; iy++) {
        for (ix = 0; ix < 16; ix++) {
            luma_pred[ix][iy] = (luma_pred_fwd[ix][iy] +
                                 luma_pred_bak[ix][iy] + 1) >> 1;
        }
    }
    for (iy = 0; iy < 8; iy++) {
        for (ix = 0; ix < 8; ix++) {
            cb_pred[ix][iy] = (cb_pred_fwd[ix][iy] +
                               cb_pred_bak[ix][iy] + 1) >> 1;
            cr_pred[ix][iy] = (cr_pred_fwd[ix][iy] +
                               cr_pred_bak[ix][iy] + 1) >> 1;
        }
    }
```

The direct mode prediction can be either progressive (see subclause 7.6.9.5) or interlaced as described below. Interlaced direct mode is used when ever the co-located macroblock (macroblock with the same coordinates) of the future reference VOP has field_predition flag is “1 Note that if the future macroblock is a skipped macroblock in a

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

P-VOP, a GMC macroblock (i.e. mcsel == '1') in an S(GMC)-VOP, or an intra macroblock, the direct mode prediction is progressive. Otherwise, interlaced direct mode prediction is used.

Interlaced direct coding mode is an extension of progressive direct coding mode. Four derived field motion vectors are calculated from the forward field motion vectors of the co-located future reference VOP, a single differential motion vector and the temporal position of the B-VOP fields with respect to the fields of the past and future reference VOPs. The four derived field motion vectors are denoted mvf[0] (top field forward) mvf[1], (bottom field forward), mvb[0] (top field backward), and mvb[1] (bottom field backward). MV[i] is the future reference picture motion vector for the top (i == 0) and bottom (i == 1) fields. Only one delta motion vector (used for both field), MVD[0], occurs in the bitstream for the field direct mode predicted macroblock. MVD[0] is decoded assuming f_code == 1 regardless of the number in VOP header. The interlaced direct mode prediction (in luma_pred[][], cb_pred[][] and cr_pred[][]) is calculated as shown below.

```
for (i = 0; i < 2; i++) {
    mvf[i].x = (TRB[i] * MV[i].x) / TRD[i] + MVD[0].x;
    mvf[i].y = (TRB[i] * MV[i].y) / TRD[i] + MVD[0].y;
    mvb[i].x = (MVD[i].x == 0) ?
        (((TRB[i] - TRD[i]) * MV[i].x) / TRD[i]) :
        mvf[i].x - MV[i].x);
    mvb[i].y = (MVD[i].y == 0) ?
        (((TRB[i] - TRD[i]) * MV[i].y) / TRD[i]) :
        mvf[i].y - MV[i].y);
field_motion_compensate_one_reference(
    luma_pred_fwd, cb_pred_fwd, cr_pred_fwd,
    luma_fwd_ref_vop, cb_fwd_ref_vop, cr_fwd_ref_vop,
    mvf[0].x, mvf[0].y, mvf[1].x, mvf[1].y,
    colocated_future_mb_top_field_reference,
    colocated_future_mb_bottom_field_reference,
    x, y, 0);
field_motion_compensate_one_reference(
    luma_pred_bak, cb_pred_bak, cr_pred_bak,
    luma_bak_ref_vop, cb_bak_ref_vop, cr_bak_ref_vop,
    mvb[1].x, mvb[1].y, mvb[1].x, mvb[1].y,
    0, 1, x, y, 0);
for (iy = 0; iy < 16; iy++) {
    for (ix = 0; ix < 16; ix++) {
        luma_pred[ix][iy] = (luma_pred_fwd[ix][iy] +
                             luma_pred_bak[ix][iy] + 1) >> 1;
    }
}
for (iy = 0; iy < 8; iy++) {
    for (ix = 0; ix < 8; ix++) {
        cb_pred[ix][iy] = (cb_pred_fwd[ix][iy] +
                           cb_pred_bak[ix][iy] + 1) >> 1;
        cr_pred[ix][iy] = (cr_pred_fwd[ix][iy] +
                           cr_pred_bak[ix][iy] + 1) >> 1;
    }
}
```

The temporal references (TRB[i] and TRD[i]) are distances in time expressed in field periods. Figure 7-48 shows how they are defined for the case where i is 0 (top field of the B-VOP). The bottom field is analogously.

--`,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

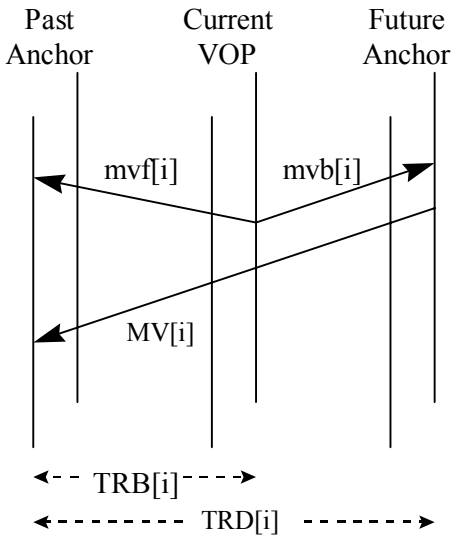


Figure 7-48 — Interlaced direct mode

The calculation of TRD[i] and TRB[i] depends not only on the current field, reference field, and frame temporal references, but also on whether the current video is top field first or bottom field first.

$$TRD[i] = 2*(T(future)//Tframe - T(past)//Tframe) + \delta[i]$$

$$TRB[i] = 2*(T(current)//Tframe - T(past)//Tframe) + \delta[i]$$

where T(future), T(current) and T(past) are the cumulative VOP times calculated from modulo_time_base and vop_time_increment of the future, current and past VOPs in display order. Tframe is the frame period determined by

$$Tframe = T(first_B_VOP) - T(past_reference_of_first\ B_VOP)$$

where first_B_VOP denotes the first B-VOP following the Video Object Layer syntax. The important thing about Tframe is that the period of time between consecutive fields which constitute an interlaced frame is assuemed to be 0.5 * Tframe for purposes of scaling the motion vectors.

The value of δ is determined from Table 7-16; it is a function of the current field parity (top or bottom), the reference field of the co-located macroblock (macroblock at the same coordinates in the furture reference VOP), and the value of top_field_first in the B-VOP's video object plane syntax.

Table 7-16 — Selection of the parameter δ

future reference VOP reference fields of the co-located macroblock		top_field_first == 0		top_field_first == 1	
Top field reference	**Bottom field reference**	**Top field, δ[0]**	**Bottom field, δ[1]**	**Top field, δ[0]**	**Bottom field, δ[1]**
0	0	0	-1	0	1
0	1	0	0	0	0
1	0	1	-1	-1	1
1	1	1	0	-1	0

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The top field prediction is based on the top field motion vector of the P-VOP macroblock of the future reference VOP. The past reference field is the reference field selected by the co-located macroblock of the future reference VOP for the top field. Analogously, the bottom field predictor is the average of pixels obtained from the future reference VOP's bottom field and the past reference field referenced by the bottom field motion vector of the corresponding macroblock of the future reference VOP. When interlaced direct mode is used, vop_time_increment_resolution must be the smallest integer greater than or equal to the number of frames per second. In each VOP, vop_time_increment counts individual frames within a second.

7.8 Sprite decoding

The subclause specifies the additional decoding process for a sprite video object. The sprite decoding can operate in two modes: basic sprite decoding and low-latency sprite decoding. Figure 7-49 is a diagram of the sprite decoding process. It is simplified for clarity.

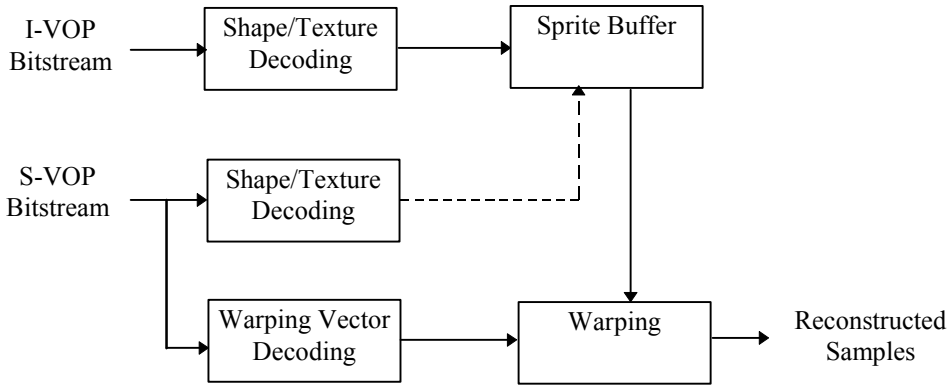


Figure 7-49 — The sprite decoding process

7.8.1 Higher syntactic structures

The various parameters in the VOL and VOP bitstreams shall be interpreted as described in Clause 6. When sprite_enable == 'static', vop_coding_type shall be "I" only for the initial VOP in a VOL for basic sprites (i.e. low_latency_sprite_enable == '0'), and all the other VOPs shall be S-VOPs (i.e. vop_coding_type == "S"). The reconstructed I-VOP in a VOL for basic sprites is not displayed but stored in a sprite memory, and will be used by all the remaining S-VOPs with sprite_enable == 'static' in the same VOL. An S-VOP with sprite_enable == 'static' is reconstructed by applying warping to the VOP stored in the sprite memory, using the warping parameters (i.e. a set of motion vectors) embedded in the VOP bitstream. Alternatively, in a VOL for low-latency sprites (i.e. low_latency_sprite_enable == '1'), these S-VOPs with sprite_enable == 'static' can update the information stored in the sprite memory before applying warping.

7.8.2 Sprite Reconstruction

The luminance, chrominance and grayscale alpha data of a sprite are stored in two-dimensional arrays. The width and height of the luminance array are specified by sprite_width and sprite_height respectively. The samples in the sprite luminance, chrominance and grayscale alpha arrays are addressed by two-dimensional integer pairs (i', j') and (i_c', j_c') as defined in the following:

- Top left luminance and grayscale alpha sample
  (i', j') = (sprite_left_coordinate, sprite_top_coordinate)

- Bottom right luminance and grayscale alpha sample
  (i', j') = (sprite_left_coordinate + sprite_width − 1,
  sprite_top_coordinate + sprite_height − 1)

- Top left chrominance sample
  (i_c', j_c') = (sprite_left_coordinate / 2, sprite_top_coordinate / 2)

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- Bottom right chrominance sample
  (i_c', j_c') = (sprite_left_coordinate / 2 + sprite_width// 2 − 1,
  sprite_top_coordinate / 2 + sprite_height// 2 − 1).

Likewise, the addresses of the luminance, chrominance and grayscale alpha samples of the VOP currently being decoded are defined in the following:

- Top left sample of luminance and grayscale alpha
  (i, j) = (0, 0) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) for non-rectangular VOPs

- Bottom right sample of luminance and grayscale alpha
  (i, j) = (video_object_layer_width - 1, video_object_layer_height - 1) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref + vop_width - 1,
  vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular VOPs

- Top left sample of chrominance
  (i_c, j_c) = (0, 0) for rectangular VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2, vop_vertical_mc_spatial_ref / 2) for non-rectangular VOPs

- Bottom right sample of chrominance
  (i_c, j_c) = (video_object_layer_width / 2 - 1, video_object_layer_height / 2 - 1) for rectangular VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2 + vop_width// 2 - 1,
  vop_vertical_mc_spatial_ref / 2 + vop_height// 2 - 1) for non-rectangular VOPs

7.8.3 Low-latency sprite reconstruction

This subclause allows a large static sprite to be reconstructed at the decoder by properly incorporating its corresponding pieces. There are two types of pieces recognized by the decoder—object and update. The decoded sprite object-piece (i.e., embedded in a S-VOP with low_latence_sprite_enable==1 and sprite_transmit_mode=="piece") is a highly quantised version of the original sprite piece while the sprite update-piece (i.e., sprite_transmit_mode=="update") is a residual designed to improve upon the quality of decoded object-piece. Sprite pieces are rectangular pieces of texture (and shape for the object-piece) and can contain "holes," corresponding to macroblocks, that do not need to be decoded. Five parameters are required by the decoder to properly incorporate the pieces: piece_quant, piece_width, piece_height, piece_xoffset, and piece_yoffset.

Macroblocks raster scanning is employed to decode each piece. However, whenever the scan encounters a macroblock which has been part of some previously sent sprite piece, then the macroblock is not decoded and its corresponding macroblock layer is empty. In that case, the decoder treats the macroblock as a hole in the current sprite piece. Since a macroblock can be refined as long as there is some available bandwidth, more than one update may be decoded per macroblock and the holes for a given refinement step have no relationship to the holes of later refinement steps. Therefore, the decoding process of a hole for an update piece is different than that for the object-piece. For the object-piece, no information is decoded at all and the decoder must "manage" where "holes" lie. (see subclause 7.8.3.1). For the update-piece, the not_coded bit is decoded to indicate whether or not one more refinement should be decoded for this given macroblock. (see subclause 7.8.3.2). Note that a hole could be non-transparent and have had shape information decoded previously. Multiple intermingled object-pieces and update-pieces may be decoded at the same current VOP. Part of a sequence could consist for example of rapidly showing a zooming out effect, a panning to the right, a zooming in, and finally a panning to the left. In this case, the first decoded object-piece covers regions on all four sides of the previous VOP transmitted piece, which is now treated as a hole and not decoded again. The second decoded object-piece relates to the right panning, and the third object-piece is a smaller left-panning piece due to the zooming-in effect. Finally, the last piece is different; instead of an object, it contains the update for some previous object-piece of zooming-in (thus, the need to update to refine for higher quality). All four pieces will be decoded within the same VOP. When sprite_transmit_mode = ="pause," the decoder recognizes that all sprite object-pieces and update-pieces for the current VOP session have been sent. However, when sprite_transmit_mode = "stop," the decoder understands that all object and update-pieces have been sent for the entire video object layer, not just for the current VOP. session. In addition, once all object-pieces or update-pieces have been decoded during a VOP session (i.e., signaled by sprite_transmit_mode == "pause" or sprite_transmit_mode == "stop"), the static sprite is padded (as defined in subclause 7.6.1), then the portion to be displayed is warped, to complete the current VOP session.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For the S-VOPs (i.e., vop_coding_type == “S”) with sprite_enable == ‘static’, the macroblock layer syntax of object-pieces is the same as those of I-VOP. Therefore, shape and texture are decoded using the macroblock layer structure in I-VOPs with the quantisation of intra macroblocks. The syntax of the update-pieces is similar to the P-VOP inter-macroblock syntax with the quantisation of non-intra macroblocks); however, the differences are indicated in Table B.1, specifically that there are no motion vectors and shape information included in this decoder syntax structure. In summary, this decoding process supports the construction of any large sprite image progressively, both spatially and in terms of quality.

7.8.3.1 Decoding of holes in sprite object-piece

Implementation of macroblock scanning must account for the possibility that a macroblock uses prediction based on some macroblock sent in a previous piece. When an object-piece with holes is decoded, the decoder in the process of reconstruction acts as if the whole original piece were decoded, but actually only the bitstream corresponding to the “new macroblock” is received. Whenever macroblocks raster scanning encounters a hole, the decoder needs to manage the retrieval of relevant information (e.g. DCT quantiser scales, AC and DC prediction parameters, and BAB bordering values) from the corresponding macroblock decoded earlier.

7.8.3.2 Decoding of holes in sprite update-pieces

In contrast to the send_mb() used by the object-pieces, the update-pieces use the not_coded bit. When not_coded = 1 in the P-VOP syntax, the decoder recognizes that the corresponding macroblock is not refined by the current sprite update-piece. When not_coded = 0 in the P-VOP syntax, the decoder recognizes that this macroblock is refined. The prediction for the update piece is obtained by extracting the "area" of the static sprite defined by (piece_width, piece_height, piece_xoffset, piece_yoffset). This area is then padded and serves as prediction for the update pieces. Since there is no shape information included in an update-piece, the result of its transparent_mb() is retrieved from the corresponding macroblock in the object-piece decoded earlier. In addition, an update macroblock cannot be transmitted before its corresponding object macroblock. As a result, the very first sprite piece transmitted in the low-latency mode shall be an object-piece.

7.8.4 Sprite reference point decoding

The syntactic elements in sprite_trajectory () and below shall be interpreted as specified in Clause 6. du[i] and dv[i] ($0 \le i <$ no_of_sprite_warping_points) specifies the mapping between indexes of some reference points in the VOP and the corresponding reference points in the sprite or reference VOP. These points are referred to as VOP reference points and sprite reference points, respectively, in the rest of the specification.

The index values for the VOP reference points are defined as:

$(i_0, j_0) = (0, 0)$ when video_object_layer_shape == ‘rectangular’, and
(vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) otherwise,
$(i_1, j_1) = (i_0+W,\ j_0)$,
$(i_2, j_2) = (i_0,\ j_0 + H)$,
$(i_3, j_3) = (i_0+W,\ j_0+H)$

where W = video_object_layer_width and H = video_object_layer_height when video_object_layer_shape == ‘rectangular’ or W = vop_width and H = vop_height otherwise. Only the index values with subscripts less than no_of_sprite_warping_points shall be used for the rest of the decoding process.

The index values for the sprite reference points shall be calculated as follows:

$(i_0', j_0') = (s / 2)\ (2\ i_0 + du[0],\ 2\ j_0 + dv[0])$
$(i_1', j_1') = (s / 2)\ (2\ i_1 + du[1] + du[0],\ 2\ j_1 + dv[1] + dv[0])$
$(i_2', j_2') = (s / 2)\ (2\ i_2 + du[2] + du[0],\ 2\ j_2 + dv[2] + dv[0])$
$(i_3', j_3') = (s / 2)\ (2\ i_3 + du[3] + du[2] + du[1] + du[0],\ 2\ j_3 + dv[3] + dv[2] + dv[1] + dv[0])$

where i_0', j_0', etc are integers in $\frac{1}{s}$ pel accuracy, where s is specified by sprite_warping_accuracy. Only the index values with subscripts less than no_of_sprite_warping_points need to be calculated.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When no_of_sprite_warping_points == 2 or 3, the index values for the *virtual sprite points* are additionally calculated as follows:

$$(i_1'', j_1'') = (16\,(i_0 + W') + ((W - W')\,(r\,i_0' - 16\,i_0) + W'\,(r\,i_1' - 16\,i_1))\ //\ W,\\ 16\,j_0 + ((W - W')\,(r\,j_0' - 16\,j_0) + W'\,(r\,j_1' - 16\,j_1))\ //\ W)$$
$$(i_2'', j_2'') = (16\,i_0 + ((H - H')\,(r\,i_0' - 16\,i_0) + H'\,(r\,i_2' - 16\,i_2))\ //\ H,\\ 16\,(j_0 + H') + ((H - H')\,(r\,j_0' - 16\,j_0) + H'\,(r\,j_2' - 16\,j_2))\ //\ H)$$

where i_1'', j_1'', i_2'', and j_2'' are integers in $\frac{1}{16}$ pel accuracy, and $r = 16/s$. W' and H' are defined as the smallest integers that satisfy the following condition:

$W' = 2^{\alpha}$, $H' = 2^{\beta}$, $W' \geq W$, $H' \geq H$, $\alpha > 0$, $\beta > 0$, both α and β are integers.

The calculation of i_2'', and j_2'' is not necessary when no_of_sprite_warping_points == 2.

7.8.5 Warping

For any pixel (i, j) inside the VOP boundary, $(F(i, j), G(i, j))$ and $(F_c(i_c, j_c), G_c(i_c, j_c))$ are computed as described in the following. These quantities are then used for sample reconstruction as specified in subclause 7.8.6. The following notations are used to simplify the description:

$$I = i - i_0,$$
$$J = j - j_0,$$
$$I_c = 4\,i_c - 2\,i_0 + 1,$$
$$J_c = 4\,j_c - 2\,j_0 + 1,$$

When no_of_sprite_warping_point == 0,

$$(F(i, j), G(i, j)) = (s\,i,\ s\,j),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (s\,i_c,\ s\,j_c).$$

When no_of_sprite_warping_point == 1 and sprite_enable == 'static,

$$(F(i, j), G(i, j)) = (i_0' + sI,\ j_0' + s\,J),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = i_0'\ ///\ 2 + s\,(i_c - i_0 / 2),\ \ j_0'\ ///\ 2 + s\,(j_c - j_0 / 2)).$$

When no_of_sprite_warping_point == 1 and sprite_enable == 'GMC',

$$(F(i, j), G(i, j)) = (i_0' + sI,\ j_0' + s\,J),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((i_0' >> 1)\ |\ (i_0'\ \&\ 1)) + s\,(i_c - i_0 / 2),\ ((j_0' >> 1)\ |\ (j_0'\ \&\ 1)) + s\,(j_c - j_0 / 2))$$

When no_of_sprite_warping_points == 2,

$$(F(i, j), G(i, j)) = (\,i_0' + ((-r\,i_0' + i_1'')\,I + (r\,j_0' - j_1'')\,J)\ ///\ (W'\,r),\\ j_0' + ((-r\,j_0' + j_1'')\,I + (-r\,i_0' + i_1'')\,J)\ ///\ (W'\,r)),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W')\ ///\ (4\,W'\,r),\\ ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2\,W'\,r\,j_0' - 16W')\ ///\ (4\,W'\,r)).$$

According to the definition of W' and H' (i.e. $W' = 2^{\alpha}$ and $H' = 2^{\beta}$), the divisions by "///" in these functions can be replaced by binary shift operations. By this replacement, the above equations can be rewritten as:

$$(F(i, j), G(i, j)) = (\,i_0' + (((-r\,i_0' + i_1'')\,I + (r\,j_0' - j_1'')\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),\\ j_0' + (((-r\,j_0' + j_1'')\,I + (-r\,i_0' + i_1'')\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),\\ ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2\,W'\,r\,j_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2)),$$

where $2^{\rho}=r$.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When no_of_sprite_warping_points == 3,

$$
\begin{aligned}
(F(i, j), G(i, j)) &= (i_0' + ((-r\, i_0' + i_1'')\, H'\, I + (-r\, i_0' + i_2'')W'\, J) /// (W'H'r),\\
&\quad\ j_0' + ((-r\, j_0' + j_1'')\, H'\, I + (-r\, j_0' + j_2'')W'\, J) /// (W'H'r)),\\
(F_c(i_c, j_c), G_c(i_c, j_c)) &= (((-r\, i_0' + i_1'')\, H'\, I_c + (-r\, i_0' + i_2'')W'\, J_c + 2\, W'H'r\, i_0' - 16W'H') /// (4W'H'r),\\
&\quad\ ((-r\, j_0' + j_1'')\, H'\, I_c + (-r\, j_0' + j_2'')W'\, J_c + 2\, W'H'r\, j_0' - 16W'H') /// (4W'H'r)).
\end{aligned}
$$

According to the definition of W' and H', the computation of these functions can be simplified by dividing the denominator and numerator of division beforehand by W' (when $W' < H'$) or H' (when $W' \geq H'$). As in the case of no_of_sprite_warping_points == 2, the divisions by "///" in these functions can be replaced by binary shift operations. For example, when $W' \geq H'$ (i.e. $\alpha \geq \beta$) the above equations can be rewritten as:

$$
\begin{aligned}
(F(i, j), G(i, j)) &= (i_0' + (((-r\, i_0' + i_1'')\, I + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta}\, J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),\\
&\quad\ j_0' + (((-r\, j_0' + j_1'')\, I + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta}\, J + 2^{\alpha+\rho-1}) >> (\alpha+\rho))),\\
(F_c(i_c, j_c), G_c(i_c, j_c)) &= (((-r\, i_0' + i_1'')\, I_c + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta}\, J_c + 2W'r\, i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),\\
&\quad\ ((-r\, j_0' + j_1'')\, I_c + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta}\, J_c + 2W'r\, j_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2)).
\end{aligned}
$$

When no_of_sprite_warping_point == 4,

$$
\begin{aligned}
(F(i, j), G(i, j)) &= ((a\, I + b\, J + c) /// (g\, I + h\, J + D\, W\, H),\\
&\quad\ (d\, I + e\, J + f) /// (g\, I + h\, J + D\, W\, H)),\\
(F_c(i_c, j_c), G_c(i_c, j_c)) &= ((2\, a\, I_c + 2\, b\, J_c + 4\, c - (g\, I_c + h\, J_c + 2\, D\, W\, H)\, s) /// (4gI_c + 4\, hJ_c + 8D\, W\, H),\\
&\quad\ (2\, d\, I_c + 2\, e\, J_c + 4\, f - (g\, I_c + h\, J_c + 2\, D\, W\, H)\, s) /// (4\, g\, I_c + 4\, hJ_c + 8D\, W\, H))
\end{aligned}
$$

where

$$
\begin{aligned}
g &= ((i_0' - i_1' - i_2' + i_3')\,(j_2' - j_3') - (i_2' - i_3')\,(j_0' - j_1' - j_2' + j_3'))\, H,\\
h &= ((i_1' - i_3')\,(j_0' - j_1' - j_2' + j_3') - (i_0' - i_1' - i_2' + i_3')\,(j_1' - j_3'))\, W,\\
D &= (i_1' - i_3')\,(j_2' - j_3') - (i_2' - i_3')\,(j_1' - j_3'),\\
a &= D\,(i_1' - i_0')\, H + g\, i_1',\\
b &= D\,(i_2' - i_0')\, W + h\, i_2',\\
c &= D\, i_0'\, W\, H,\\
d &= D\,(j_1' - j_0')\, H + g\, j_1',\\
e &= D\,(j_2' - j_0')\, W + h\, j_2',\\
f &= D\, j_0'\, W\, H.
\end{aligned}
$$

A set of parameters that causes the denominator of any of the above equations to be zero for any pixel in an opaque or boundary macroblock is disallowed. The implementor should be aware that a 32 bit register may not be sufficient for representing the denominator or the numerator in the above transform functions for affine and perspective transform. The usage of a 64 bit floating point representation should be sufficient in such case.

7.8.6 Sample reconstruction

The reconstructed value Y of the luminance sample (i, j) in the currently decoded VOP shall be defined as

$$Y = ((s - r_j)((s - r_i)\, Y_{00} + r_i\, Y_{01}) + r_j\,((s - r_i)\, Y_{10} + r_i\, Y_{11})) // s^2,$$

when sprite_enable == 'static', and

$$Y = ((s - r_j)((s - r_i)\, Y_{00} + r_i\, Y_{01}) + r_j\,((s - r_i)\, Y_{10} + r_i\, Y_{11}) + s^2/2 - \text{rounding_control}) / s^2,$$

when sprite_enable == 'GMC', where Y_{00}, Y_{01}, Y_{10}, Y_{11} represent the luminance sample in the sprite or reference VOP at $(F(i, j)////s, G(i, j)////s)$, $(F(i, j)////s + 1, G(i, j)////s)$, $(F(i, j)////s, G(i, j)////s + 1)$, and $(F(i, j)////s + 1, G(i, j)////s + 1)$ respectively, and $r_i = F(i, j) - (F(i, j)////s)s$ and $r_j = G(i, j) - (G(i, j)////s)s$. See sublause 7.6.2 for the definition of rounding_control. Figure 7-50 illustrates this process.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In case any of Y_{00}, Y_{01}, Y_{10} and Y_{11} lies outside the luminance binary mask of the sprite or reference VOP, it shall be obtained by the padding process as defined in subclause 7.6.1. Additionally, in the case when the current VOP is a rectangular S(GMC)-VOP and any of Y_{00}, Y_{01}, Y_{10} and Y_{11} lies outside the luminance bounding rectangle of the reference VOP, the reconstructed sample value is obtained as described in subclause 7.6.4.

When brightness_change_in_sprite == 1, the final reconstructed luminance sample (i, j) is further computed as Y = *(Y* * (brightness_change_factor + 100)) // 100, clipped to the range of [0, $2^{bits_per_pixel}$-1].

Similarly, the reconstructed value C of the chrominance sample (i_c, j_c) in the currently decoded VOP shall be defined as

$$C = ((s - r_j)((s - r_i)\, C_{00} + r_i\, C_{01}) + r_j\, ((s - r_i)\, C_{10} + r_i\, C_{11})) \,//\, s^2,$$

when sprite_enable == 'static', and

$$C = ((s - r_j)((s - r_i)\, C_{00} + r_i\, C_{01}) + r_j\, ((s - r_i)\, C_{10} + r_i\, C_{11}) + s^2/2 - \text{rounding_control}) \,/\, s^2,$$

when sprite_enable == 'GMC', where C_{00}, C_{01}, C_{10}, C_{11} represent the chrominance sample in the sprite or reference VOP at ($F_c(i_c, j_c)$////s, $G_c(i_c, j_c)$////s), ($F_c(i_c, j_c)$////s + 1, $G_c(i_c, j_c)$////s), ($F_c(i_c, j_c)$////s, $G_c(i_c, j_c)$////s + 1), and ($F_c(i_c, j_c)$////s + 1, $G_c(i_c, j_c)$////s + 1) respectively, and $r_i = F_c(i_c, j_c) - (F_c(i_c, j_c))$////$s)s$ and $r_j = G_c(i_c, j_c) - (G_c(i_c, j_c)$////$s)s$. In case any of C_{00}, C_{01}, C_{10} and C_{11} lies outside the chrominance binary mask of the sprite or reference VOP, it shall be obtained by the padding process as defined in subclause 7.6.1. Additionally, in the case when the current VOP is a rectangular S(GMC)-VOP and any of C_{00}, C_{01}, C_{10} and C_{11} lies outside the chrominance bounding box of the reference VOP, the reconstructed sample value is obtained as described in subclause 7.6.4.

The same method is used for the reconstruction of grayscale alpha and luminance samples. The reconstructed value A of the grayscale alpha sample (i, j) in the currently decoded VOP shall be defined as

$$A = ((s - r_j)((s - r_i)\, A_{00} + r_i\, A_{01}) + r_j\, ((s - r_i)\, A_{10} + r_i\, A_{11})) \,//\, s^2,$$

when sprite_enable == 'static', and

$$A = ((s - r_j)((s - r_i)\, A_{00} + r_i\, A_{01}) + r_j\, ((s - r_i)\, A_{10} + r_i\, A_{11}) + s^2/2 - \text{rounding_control}) \,/\, s^2,$$

when sprite_enable == 'GMC', where A_{00}, A_{01}, A_{10}, A_{11} represent the grayscale alpha samples in the sprite or reference VOP at (F(i, j)////s, G(i, j)////s), (F(i, j)////s + 1,G(i, j)////s), (F(i, j)////s, G(i, j)////s + 1), and (F(i, j)////s + 1,G(i, j)////s + 1) respectively, and $r_i = F(i, j) - (F(i, j)$////$s)s$ and $r_j = G(i, j) - (G(i, j)$////$s)s$. In case any of A_{00}, A_{01}, A_{10} and A_{11} lies outside the luminance binary mask of the sprite or reference VOP, it shall be obtained by the padding process as defined in subclause 7.6.1. Additionally, in the case when the current VOP is a rectangular S(GMC)-VOP and any of A_{00}, A_{01}, A_{10} and A_{11} lies outside the grayscale alpha bounding box of the reference VOP, the reconstructed sample value is obtained as described in subclause 7.6.4.

When sprite_enable == "static", the reconstructed value of luminance binary mask sample BY(i,j) shall be computed following the identical process for the luminance sample. However, corresponding binary mask sample values shall be used in place of luminance samples Y_{00}, Y_{01}, Y_{10}, Y_{11}. Assume the binary mask sample opaque is equal to 255 and the binary mask sample transparent is equal to 0. If the computed value is bigger or equal to 128, BY(i, j) is defined as opaque. Otherwise, BY (i, j) is defined as transparent. The chrominance binary mask samples shall be reconstructed by decimating of the corresponding 2 x 2 adjacent luminance binary mask samples as specified in subclause 7.6.1.4.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

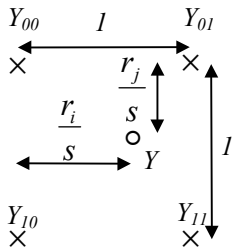


Figure 7-50 — Pixel value interpolation (it is assumed that sprite samples are located on an integer grid)

7.8.7 GMC decoding

7.8.7.1 GMC prediction

For S(GMC)-VOPs, the pixels in a macroblock (including pixels in the grayscale alpha MB) are predicted using GMC prediction when mcsel == '1'. The predicted macroblock is obtained by applying warping to the reference VOP. The prediction error is decoded by the same method as in usual interframe motion compensation. The prediction based on warping is performed using sprite reference point decoding described in subclause 7.8.4, sub-pel warping described in subclause 7.8.5, and sample reconstruction described in subclause 7.8.6.

7.8.7.2 Texture and shape decoding in S(GMC)-VOP

Decoding of texture and shape in S(GMC)-VOPs follows the same rule as those in P-VOPs. Global warping parameters are not used for shape decoding in S(GMC)-VOPs. Macroblocks with mcsel == '1' are regarded as intra-coded macroblocks in shape motion vector decoding since they do not have their own motion vector.

Overlapped block MC is disabled over the boundary between macroblocks with different values for mcsel.

When interlaced == '1', macroblocks with mcsel == "1" are predicted using frame prediction (i.e. the warping method is identical with the case when interlaced == '0').

7.8.7.3 Motion vector decoding

Motion vectors of macroblocks with mcsel == '0' and not_coded == '0' in S-VOPs are decoded using the same rules as P-VOPs. Although macroblocks with mcsel == '1' do not have their own *block* motion vectors, they have *pel-wise* motion vectors for sprite warping obtained from global motion parameters. The candidate motion vector predictor from the reference macroblock with mcsel == '1' is obtained as the *averaged value* of the pel-wise motion vectors in the macroblock. Note that any reference macroblock outside video packet is treated as transparent. In case the frame-based direct mode in B-VOPs, the motion vector of the co-located macroblock with mcsel == '1' is also obtained as the averaged value of the pel-wise motion vectors in the macroblock when the most recently decoded VOP is S(GMC)-VOP.

Note : Averaged value of pel-wise vectors is calculated using the all luminance pixels in the macroblock, that is 256 pixels are used (Nb is always 256 into the following expression).

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

$$AMV_x = \left(\sum_y \sum_x MV_x(x,y) \right) / N_b$$

$$AMV_y = \left(\sum_y \sum_x MV_y(x,y) \right) / N_b$$

MVx(x,y): Horizontal motion vector at *(x,y)*

MVy(x,y): Vertical motion vector at *(x,y)*

N_b: Number of pixels in the reference block

AMVx: Averaged value of horizontal motion vectors in the block

AMVy: Averaged value of vertical motion vectors in the block

Here,

Since the *AMVs* have fractional values, they are quantised to half integer values when quarter_sample == '0', and quarter integer values when quarter_sample == '1', using the "//" operator. For example, values within [0.0, 0.25) are rounded to 0, [0.25, 0.75) are rounded to 0.5, and [0.75, 1.0] are rounded to 1.0 when quarter_sample == '0'. Values within [0.0, 0.125) are rounded to 0, [0.125, 0.375) are rounded to 0.25, [0.375, 0.625) are rounded to 0.5, [0.625, 0.875) are rounded to 0.75, and [0.875, 1.0] are rounded to 1.0 when quarter_sample == '1'.

If the quantised AMV is outside the motion vector range specified by f_code, it is clipped in the range.

The operation above is performed independently for horizontal and vertical components.

For example, if the left block is coded in GMC (Figure 7-51), the candidate predictor is obtained as the averaged value of the pel-wise motion vectors in the left block.

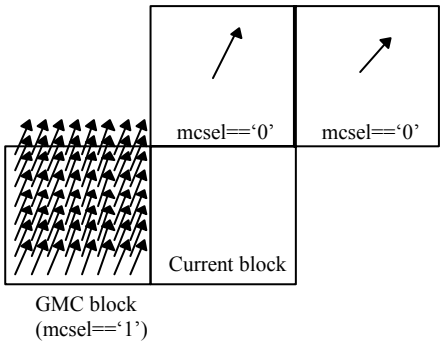


Figure 7-51 — Pel-wise motion vectors of GMC blocks

7.9 Generalized scalable decoding

This subclause specifies the additional decoding process required for decoding scalable coded video.

The scalability framework is referred to as generalized scalability which includes the spatial and the temporal scalabilities. The temporal scalability offers scalability of the temporal resolution, and the spatial scalability offers scalability of the spatial resolution. Each type of scalability involves more than one layer. In the case of two layers, consisting of a lower layer and a higher layer; the lower layer is referred to as the reference layer and the higher layer is called the enhancement layer.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the case of temporal scalability, both rectangular VOPs as well as arbitrary shaped VOPs are supported. In the case of spatial scalability, only rectangular VOPs are supported. Figure 7-52 shows a high level decoder structure for generalized scalability.

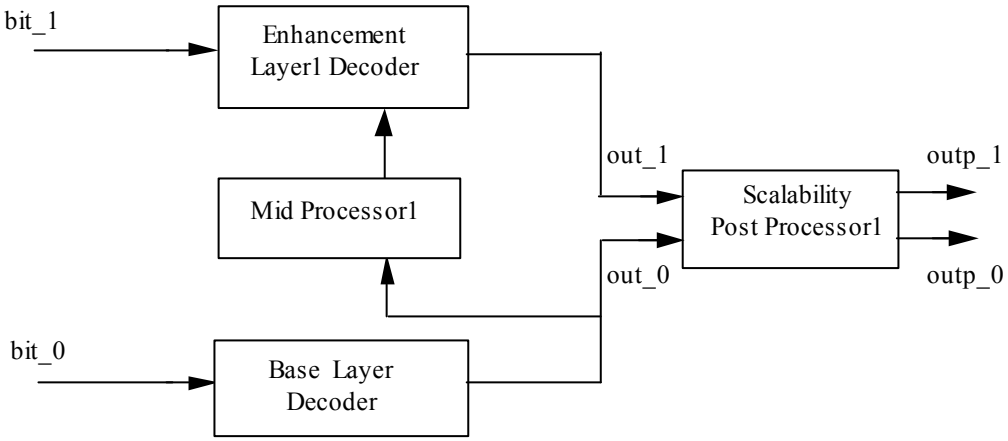


Figure 7-52 — High level decoder structure for generalized scalability

The base layer and enhancement layer bitstreams are input for decoding by the corresponding base layer decoder and enhancement layer decoder.

When spatial scalability is to be performed, mid processor 1 performs spatial up or down sampling of input. The scalability post processor performs any necessary operations such as spatial up or down sampling of the decoded reference layer for display resulting at outp_0 while the enhancement layer without resolution conversion may be output as outp_1.

When temporal scalability is to be performed, the decoding of base and enhancement layer bitstreams occurs in the corresponding base and enhancement layer decoders as shown. In this case, mid processor 1 does not perform any spatial resolution conversion. The post processor outputs the base layer VOPs without any conversion, but temporally multiplexes the base and enhancement layer VOPs to produce higher temporal resolution enhancement layer.

The reference VOPs for prediction are selected by ref_select_code as specified in Table 7-17 and Table 7-18. In coding of P-VOPs belonging to an enhancement layer, the forward reference is one of the following four: the most recently decoded VOP of enhancement layer, the most recent VOP of the reference layer in display order, the next VOP of the reference layer in display order, or the temporally coincident VOP in the reference layer.

In B-VOPs, the forward reference is one of the following two: the most recently decoded enhancement VOP or the most recent reference layer VOP in display order. The backward reference is one of the following three: the temporally coincident VOP in the reference layer, the most recent reference layer VOP in display order, or the next reference layer VOP in display order.

In case of hierarchy_type == 1(temporal scalability), the default value of ref_select_code is “00”. The value of vop_coded should be “1” for the first VOP in an enhancement layer if the value of video_object_layer_shape is ‘00’.

In case of hierarchy_type == 0(spatial scalability), the default value of ref_select_code is “11” for P-VOP and "00" for B-VOP. The value of vop_coded should be “1” for the first VOP in an enhancement layer if the value of video_object_layer_shape is ‘00’.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-17 — Prediction reference choices in enhancement layer P-VOPs for scalability

ref_select_code	forward prediction reference
00	Most recently decoded enhancement VOP belonging to the same layer for which “vop_coded == 1”.
01	Most recently VOP in display order belonging to the reference layer for which “vop_coded == 1”.
10	Next VOP in display order belonging to the reference layer.
11	Temporally coincident VOP in the reference layer (no motion vectors)

NOTE 1 — The value “10” is prohibited if “vop_coded = 0” for the next VOP in display order belonging to the reference layer.

NOTE 2 — The value “11” is prohibited if “vop_coded = 0” for the temporally coincident VOP in the reference layer.

Table 7-18 — Prediction reference choices in enhancement layer B-VOPs for scalability

ref_select_code	forward temporal reference	backward temporal reference
00	Most recently decoded enhancement VOP of the same layer for which “vop_coded == 1”.	Temporally coincident VOP in the reference layer (no motion vectors)
01	Most recently decoded enhancement VOP of the same layer for which “vop_coded == 1”.	Most recent VOP in display order belonging to the reference layer.
10	Most recently decoded enhancement VOP of the same layer for which “vop_coded == 1”.	Next VOP in display order belonging to the reference layer.
11	Most recent VOP in display order belonging to the reference layer for which “vop_coded == 1”.	Next VOP in display order belonging to the reference layer.

The following combinations of video_object_layer_shape in the base and enhancement layer and enhancement_type in the enhancement_layer are supported.

	Base Layer	Enhancement Layer	Enhancement _type	status
hierarchy_type=1	rectangular	rectangular	0	supported
	rectangular	binary shape	1	supported
	binary Shape	binary shape	0/1	supported
	binary Shape	rectangular	-	forbidden
	binary only	binary only	-	forbidden
hierarchy_type=0 && video_object_layer_ver_id = 0001	rectangular	rectangular	0	supported
	rectangular	binary shape	-	forbidden
	binary shape	binary shape	-	forbidden
	binary shape	rectangular	-	forbidden
	binary only	binary only	-	forbidden
hierarchy_type=0 && video_object_layer_ver_id = 0010	rectangular	rectangular	0	supported
	rectangular	binary shape	1	supported
	binary shape	binary shape	0/1	supported
	binary shape	rectangular	-	forbidden
	binary only	binary only	-	supported

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.1 Temporal scalability

Temporal scalability involves two layers, a lower layer and an enhancement layer. Both the lower and the enhancement layers process the same spatial resolution. The enhancement layer enhances the temporal resolution of the lower layer and if temporally remultiplexed with the lower layer provides full temporal rate.

7.9.1.1 Base layer and enhancement layer

In the case of temporal scalability, the decoded VOPs of the enhancement layer are used to increase the frame rate of the base layer. Figure 7-53 shows a simplified diagram of the motion compensation process for the enhancement layer using temporal scalability.

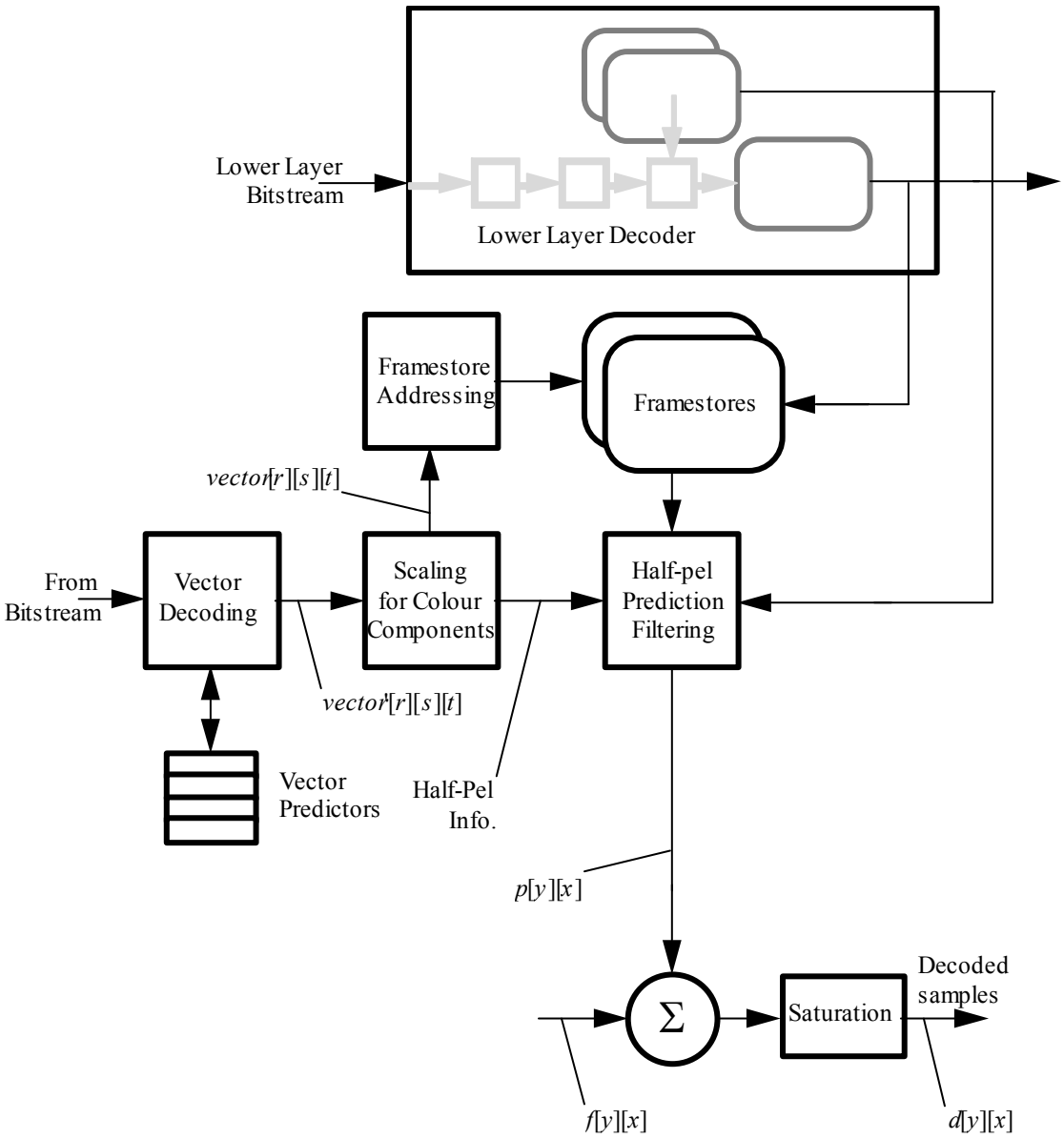


Figure 7-53 — Simplified motion compensation process for temporal scalability

Predicted samples p[y][x] are formed either from frame stores of the reference layer or from frame stores of enhancement layer. The difference data samples f[y][x] are added to p[y][x] to form the decoded samples d[y][x].

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

There are two types of enhancement structures indicated by the “enhancement_type” flag. When the value of enhancement_type is “1”, the enhancement layer increases the temporal resolution of a partial region of the reference layer. When the value of enhancement_type is “0”, the enhancement layer increases the temporal resolution of an entire region of the reference layer.

The value of video_object_layer_width in base layer and enhancement layer shall be identical if both layers are rectangular. The value of video_object_layer_height in base layer and enhancement layer shall be identical if both layers are rectangular.

7.9.1.2 Base layer

The decoding process of the base layer is the same as non-scalable decoding process.

7.9.1.3 Enhancement layer

The VOP of the enhancement layer is decoded as either I-VOP, P-VOP or B-VOP. The shape of the VOP is either rectangular (video_object_layer_shape is “00”) or arbitrary (video_object_layer_shape is “01”). B-VOP in base layer shall not be used as a reference for enhancement layer VOP although B-VOP in enhancement layer can be a reference for enhancement layer VOP.

7.9.1.3.1 Decoding of I-VOPs

The decoding process of the I-VOPs in the enhancement layer is the same as the non-scalable decoding process. ref_layer_id, ref_layer_sampling_direc, hor_sampling_factor_n, horsampling_factor_m, vertical_sampling_factor_n, vertical_sampling_factor_m and ref_select_code are ignored in the temporal scalability I-VOPs.

7.9.1.3.2 Decoding of P-VOPs

The reference layer is indicated by ref_layer_id in Video Object Layer class. Other decoding process is the same as non-scalable P-VOPs except the process specified in subclauses 7.9.1.3.4 and 7.9.1.3.5.

For P-VOPs, the ref_select_code is either “00”, “01” or “10”.

When the value of ref_select_code is “00”, the prediction reference is set by the most recently decoded VOP belonging to the same layer.

When the value of ref_select_code is “01”, the prediction reference is set by the previous VOP in display order belonging to the reference layer.

When the value of ref_select_code is “10”, the prediction reference is set by the next VOP in display order belonging to the reference layer.

In the case of hierarchy_type=1(temporal scalability), hor_sampling_factor_n, hor_sampling_factor_m, vert_sampling_factor_n and vert_sampling_factor_m are ignored.

7.9.1.3.3 Decoding of B-VOPs

The reference layer is indicated by ref_layer_id in Video Object Layer class. Other decoding process is the same as non-scalable B-VOPs except the process specified in subclauses 7.9.1.3.4 and 7.9.1.3.5.

For B-VOPs, the ref_select_code is either “01”, “10” or “11”.

When the value of ref_select_code is “01”, the forward prediction reference is set by the most recently decoded VOP belonging to the same layer and the backward prediction reference is set by the previous VOP in display order belonging to the reference layer.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the value of ref_select_code is “10”, the forward prediction reference is set by the most recently decoded VOP belonging to the same layer, and the backward prediction reference is set by the next VOP in display order belonging to the reference layer.

When the value of ref_select_code is “11”, the forward prediction reference is set by the previous VOP in display order belonging to the reference layer and the backward prediction reference is set by the next VOP in display order belonging to the reference layer. The picture type of the reference VOP shall be either I or P (vop_coding_type = “00” or “01”).

When the value of ref_select_code is “01” or “10”, direct mode is not allowed. modb shall always exist in each macroblock, i.e. the macroblock is not skipped even if the co-located macroblock is skipped.

In the case of hierarchy_type=1(temporal scalability), hor_sampling_factor_n, hor_sampling_factor_m, vert_sampling_factor_n and vert_sampling_factor_m are ignored.

7.9.1.3.4 Decoding of arbitrary shaped VOPs

The vop_shape_coding_type for enhancement layer is ’0’ when vop_coding_type is ‘00’. Otherwise, the vop_shape_coding_type for enhancement layer is ’1’. Prediction for arbitrary shape in P-VOPs or in B-VOP is same as the one in the base layer (see subclause 7.5.2.1.2).

For arbitrary shaped VOPs with the value of enhancement_type being “1”, the shape of the reference VOP is defined as an all opaque rectangle whose size is the same as the reference layer when the shape of the reference layer is rectangular (video_object_layer_shape = “00”).

When the value of ref_select_code is “11” and the value of enhancement_type is “1”, modb shall always exist in each macroblock, i.e. the macroblock is not skipped even if the co-located macroblock is skipped.

7.9.1.3.5 Decoding of backward and forward shape

Backward shape and forward shape are used in the background composition process specified in subclause 8.1. The backward shape is the shape of the enhanced object at the next VOP in display order belonging to the reference layer. The forward shape is the shape of the enhanced object at the previous VOP in display order belonging to the reference layer.

For the VOPs with the value of enhancement_type being “1”, backward shape is decoded when the load_backward_shape is “1” and forward shape is decoded when load_forward_shape is “1”.

When the value of load_backward_shape is “1” and the value of load_forward_shape is “0”, the backward shape of the previous VOP is copied to the forward shape for the current VOP. When the value of load_backward_shape is “0”, the backward shape of the previous VOP is copied to the backward shape for the current VOP and the forward shape of the previous VOP is copied to the forward shape for the current VOP.

The decoding process of backward and forward shape is the same as the decoding process for the shape of I-VOP with binary only mode (video_object_layer_shape = “10”).

7.9.2 Spatial scalability

7.9.2.1 Binary shape spatial scalability

Refer to the subclause 7.5.4.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.2.2 Texture Spatial Scalability

7.9.2.3 Base Layer and Enhancement Layer

In the case of spatial scalability, the enhancement bitstream is used to increase the resolution of the image. When the output with the lowest resolution is required, only the base layer is decoded. When the output with higher resolution is required, both the base layer and the enhancement layer are decoded.

Figure 7-54 is a diagram of the video decoding process with spatial scalability.

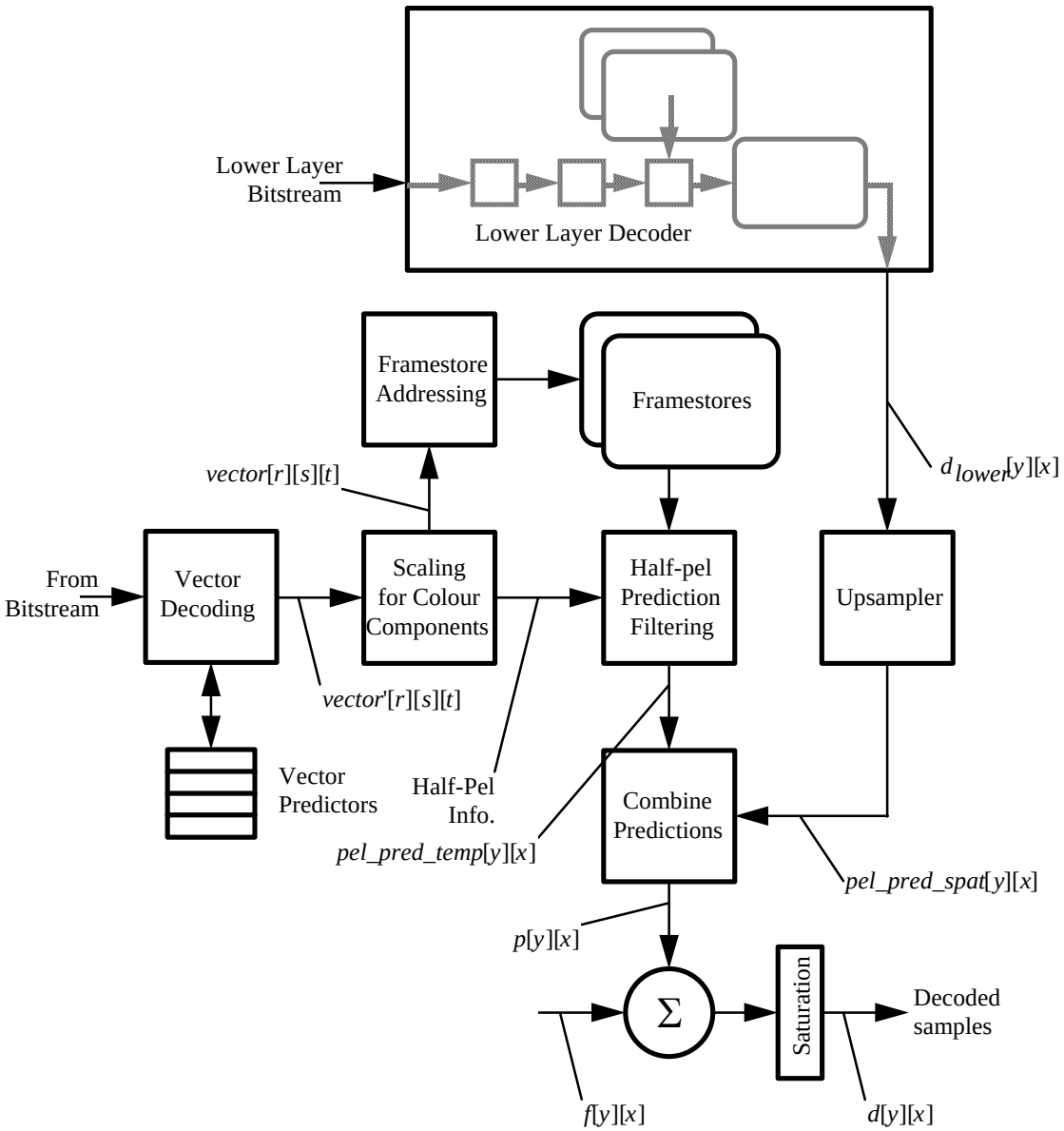


Figure 7-54 — Simplified motion compensation process for spatial scalability

7.9.2.4 Decoding of Base Layer

The decoding process of the base layer is the same as nonscalable decoding process.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.2.5 Prediction in the enhancement layer

A motion compensated temporal prediction is made from reference VOPs in the enhancement layer. In addition, a spatial prediction is formed from the lower layer decoded frame ($d_{lower}[y][x]$). These predictions are selected individually or combined to form the actual prediction.

In the enhancement layer, the forward prediction in P-VOP and the backward prediction in B-VOP are used as the spatial prediction. The reference VOP is set to the temporally coincident VOP in the reference layer. The forward prediction in B-VOP is used as the temporal prediction from the enhancement layer VOP. The reference VOP is set to the most recently decoded VOP of the enhancement layer. The interpolated prediction in B-VOP is the combination of these predictions. If the reference VOP has arbitrary shape, padding process shall be done before resampling.

In the case that a macroblock is not coded, either because the entire macroblock is skipped or the specific macroblock is not coded there is no coefficient data. In this case f[y][x] is zero, and the decoded samples are the prediction, p[y][x].

7.9.2.6 Formation of spatial prediction

Forming the spatial prediction requires definition of the spatial resampling process. The formation is performed at the mid-processor. The resampling process is defined for a whole VOP, however, for decoding of a macroblock, only the 16x16 region in the upsampled VOP, which corresponds to the position of this macroblock, is needed.

The spatial prediction is made by resampling the reconstructed VOP in the reference layer to the same sampling grid as the enhancement layer. In the first step, the reconstructed VOP is subject to vertical resampling. Then, the vertically resampled image is subject to horizontal resampling. If the reference VOP in the reference lower layer has rectangular shape, the reference VOP is re-sampled without padding process. If the reference VOP in the reference lower layer has arbitrary shape, padding process shall be done before the re-sampling process. The padding process in the formation of spatial prediction is identical to the padding process in the base layer, which is described in subclause 7.5.

In the case of arbitrary shape VOPs, its size and locations change from time to time. In order to ensure the formation of the spatial prediction, it is necessary to identify the location of the resampled VOPs in the absolute coordinate system. If the reference layer is rectangular in shape, the spatial reference of the resampled VOP is set to (0,0).

In case of arbitrary shaped VOP, the VOP location of each layer should conserve the absolute location as described in Figure 7-55.

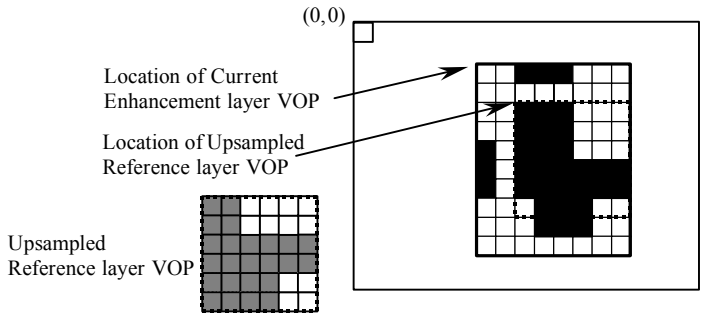


Figure 7-55 — VOP location of each layer

7.9.2.6.2 Vertical resampling

The image subject to vertical resampling, $d_{lower}[y][x]$, is resampled to the enhancement layer vertical sampling grid using linear interpolation between the sample sites according to the following formula, where vert_pic is the resulting image:

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

vert_pic[y_h][x] = (16 - phase) * d_{lower} [y1][x] + phase * d_{lower} [y2][x]
where
y_h = output sample coordinate in vert_pic
y1 = (y_h * vertical_sampling_factor_m) / vertical_sampling_factor_n
y2 = y1 + 1 if y1 < video_object_layer_height - 1
y1 otherwise
phase = (16 * ((y_h * vertical_sampling_factor_m) %
vertical_sampling_factor_n)) // vertical_sampling_factor_n

where video_object_layer_height is the height of the reference VOL.

If the reference layer VOP is arbitrary shaped VOP, the following definition should be used for y1 and y2.

y1 = MIN((MAX((y_h * vertical_sampling_factor_m) / vertical_sampling_factor_n,
VOP_vertical_mc_spatial_ref)), VOP_vertical_mc_spatial_ref + vop_height-1)
y2 = MIN((MAX((y_h * vertical_sampling_factor_m) / vertical_sampling_factor_n +1,
VOP_vertical_mc_spatial_ref)), VOP_vertical_mc_spatial_ref + vop_height-1)

where vop_height is the height of the reference VOP.

Samples which lie outside the vertically upsampled reconstructed frame which are required for upsampling are obtained by border extension of the vertically upsampled reconstructed frame.

NOTE In case of rectangular shape, the calculation of phase assumes that the sample position in the enhancement layer at yh = 0 is spatially coincident with the first sample position of the lower layer. It is recognised that this is an approximation for the chrominance component if the chroma_format == 4:2:0.

7.9.2.6.3 Horizontal resampling

The image subject to horizontal resampling, $vert_pict[y][x]$, is resampled to the enhancement layer horizontal sampling grid using linear interpolation between the sample sites according to the following formula, where hor_pic is the resulting image:

hor_pic[y][xh] = ((16 - phase) * vert_pic[y][x1] + phase * vert_pic[y][x2]) // 256
where
xh = output sample coordinate in hor_pic
x1 = (xh * horizontal_sampling_factor_m) / horizontal_sampling_factor_n
x2 = x1 + 1 if x1 < video_object_layer_width - 1
x1 otherwise
phase = (16 * ((xh * horizontal_sampling_factor_m) %
horizontal_sampling_factor_n)) // horizontal_sampling_factor_n

where video_object_layer_width is the width of the reference VOL.

If the reference layer VOP is arbitrary shaped VOP, the following definition should be used for x1 and x2.

x1 = MIN((MAX((x_h * horizontal_sampling_factor_m) / horizontal_sampling_factor_n,
vop_horizontal_mc_spatial_ref)), vop_horizontal_mc_spatial_ref + vop_width-1)
x2 = MIN((MAX((x_h * horizontal_sampling_factor_m) / horizontal_sampling_factor_n +1,
vop_horizontal_mc_spatial_ref)), vop_horizontal_mc_spatial_ref + vop_width-1)

where vop_width is the width of the reference VOP.

Samples which lie outside the lower layer reconstructed frame which are required for upsampling are obtained by border extension of the lower layer reconstructed frame.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.9.2.7 Selection and combination of spatial and temporal predictions

The spatial and temporal predictions can be selected or combined to form the actual prediction in B-VOP. The spatial prediction is referred to as “backward prediction”, while the temporal prediction is referred to as “forward prediction”. The combination of these predictions can be used as “interpolate prediction”. In the case of P-VOP, only the spatial prediction (prediction from the reference layer) can be used as the forward prediction. The prediction in the enhancement layer is defined in the following formulae.

```
pel_pred[y][x] = pel_pred_temp[y][x]                          (forward in B-VOP)
pel_pred[y][x] = pel_pred_spat[y][x] = hor_pict[y][x]         (forward in P-VOP and backward in B-VOP)
pel_pred[y][x] = (pel_pred_temp[y][x] + pel_pred_spat[y][x])//2   (Interpolate in B-VOP)
```

pel_pred_temp[y][x] is used to denote the temporal prediction (formed within the enhancement layer). pel_pred_spat[y][x] is used to denote the prediction formed from the lower layer. pel_pred[y][x] is denoted the resulting prediction.

7.9.2.8 Decoding process of enhancement layer

The VOP in the enhancement layer is decoded as either I-VOP, P-VOP or B-VOP.

7.9.2.8.1 Decoding of I-VOPs

The decoding process of the I-VOPs in the enhancement layer is the same as the non_scalable decoding process. ref_layer_id, ref_layer_sampling_direc, hor_sampling_factor_n, horsampling_factor_m, vertical_sampling_factor_n, vertical_sampling_factor_m and ref_select_code are ignored in the spatial scalability I-VOPs.

7.9.2.8.2 Decoding of P-VOPs

In P-VOP, the ref_select_code shall be “11”, i.e., the prediction reference is set to the temporally coincident VOP in the reference layer. The reference layer is indicated by ref_layer_id in VideoObjectLayer class. In the case of spatial prediction, the motion vector shall be set to 0 at the decoding process and is not encoded in the bitstream.

A variable length codeword giving information about the macroblock type and the coded block pattern for chrominance is mcbpc. The codewords for mcbpc in the enhancement layer are the same as the base layer and shown in Table B.7. mcbpc shall be included in coded macroblocks.

The macroblock type gives information about the macroblock and which data elements are present. Macroblock types and included elements in the enhancement layer bitstream are listed in subclause B.1.1.

In the case of the enhancement layer of spatial scalability, INTER4V shall not be used. The macroblock of INTER or INTER+Q is encoded using the spatial prediction.

7.9.2.8.3 Decoding of B-VOPs

In B-VOP, the ref_select_code shall be “00”, i.e., the backward prediction reference is set to the temporally coincident VOP in the reference layer, and the forward prediction reference is set to the most recently decoded VOP in the enhancement layer. In the case of spatial prediction, the motion vector shall be set to 0 at the decoding process and is not encoded in the bitstream.

modb shall be present in coded macroblocks belonging to B-VOPs. The codeword is the same as the base layer and is shown in Table B.3. In case mb_type does not exist the default shall be set to "Forward MC" (prediction from the last decoded VOP in the same reference layer). modb shall be encoded in all macroblocks. If its value is equal to ‘1’, further information is not transmitted for this macroblock. The decoder treats the prediction of this macroblock as forward MC with motion vector equal to zero.

mb_type is present only in coded macroblocks belonging to B-VOPs. The mb_type gives information about the macroblock and which data elements are present. mb_type and included elements in the enhancement layer bitstream are listed in Table B.5.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the case of the enhancement layer of spatial scalability, direct mode shall not be used. The decoding process of the forward motion vectors are the same as the base layer.

7.9.2.9 Decoding of binary shape

The decoding process of the binary shape is the same as binary shape spatial scalable decoding process described in sublcause 7.5.4.

7.10 Still texture object decoding

The block diagram of the decoder is shown in Figure 7-56.

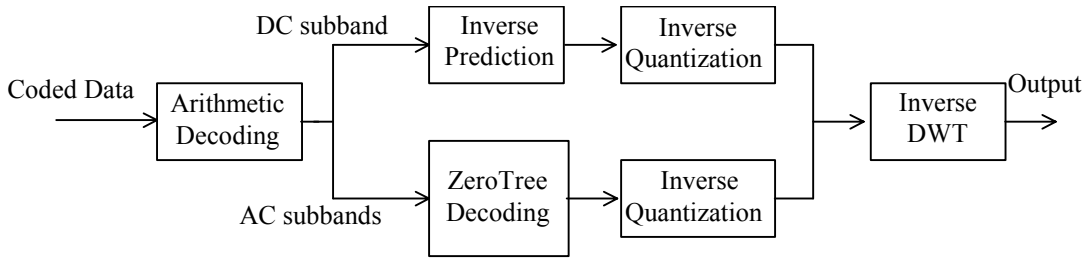


Figure 7-56 — Block diagram of the decoder

The basic modules of a zero-tree wavelet based decoding scheme are as follows:

Arithmetic decoding of the DC subband using a predictive scheme.
Arithmetic decoding of the bitstream into quantised wavelet coefficients and the significance map for AC subbands.
Zero-tree decoding of the higher subband wavelet coefficients.
Inverse quantisation of the wavelet coefficients.
Composition of the texture using inverse discrete wavelet transform (IDWT).

7.10.1 Decoding of the DC subband

The wavelet coefficients of DC subband are decoded independently from the other bands. First the quantisation step size decoded, then the offset value of the quantisation indices “band_offset” and the maximum value of the differential quantisation indices “band_max_value” are decoded from bitstream. The parameter “band_offset” is negative or zero integer and the parameter “band_max_value” is a positive integer, so only the magnitude of these parameters are read from the bitstream.

The arithmetic model is initialized with a uniform distribution. Then, the differential quantisation indices are decoded using the arithmetic decoder in a bitplane-by-bitplane fashion, from MSB to LSB. In each bitplane, the indices are decoded in a raster scan order, starting from the upper left index and ending with the lowest right one. Separate probability model for each bitplane and color component is used. The model is updated with the decoding of each bits of the predicted wavelet quantisation index to adopt the probability model to the statistics of DC band.

The “band_offset” is added to all the decoded quantisation indices, and an inverse predictive scheme is applied. Each of the current indices w_X is predicted from three quantisation indices in its neighborhood, i.e. w_A, w_B, and w_C (see Figure 7-57), and the predicted value is added to the current decoded coefficient. That is,

if $(|w_A - w_B|) < |w_B - w_C|)$
$\hat{w}_x = w_C$
else
$\hat{w}_x = w_A$
$w_x = w_x + \hat{w}_x$

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If any of nodes A, B or C is not in the image, its value is set to zero for the purpose of the inverse prediction. Finally, the inverse quantisation scheme is applied to all decoded values to obtain the wavelet coefficients of DC band.

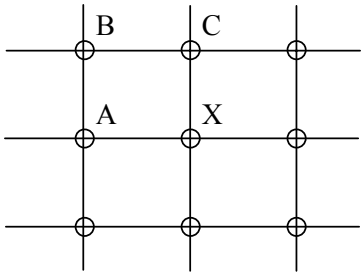


Figure 7-57 — DPCM decoding of DC band coefficients

7.10.2 ZeroTree Decoding of the Higher Bands

The zero-tree algorithm is based on the strong correlation between the amplitudes of the wavelet coefficients across scales, and on the idea of partial ordering of the coefficients. The coefficient at the coarse scale is called the *parent*, and all coefficients at the same spatial location, and of similar orientation, at the next finer scale are that parent's children. Figure 7-58 shows a wavelet tree where the parents and the children are indicated by squares and connected by lines. Since the DC subband (shown at the upper left in Figure 7-58) is coded separately using a DPCM scheme, the wavelet trees start from the adjacent higher bands.

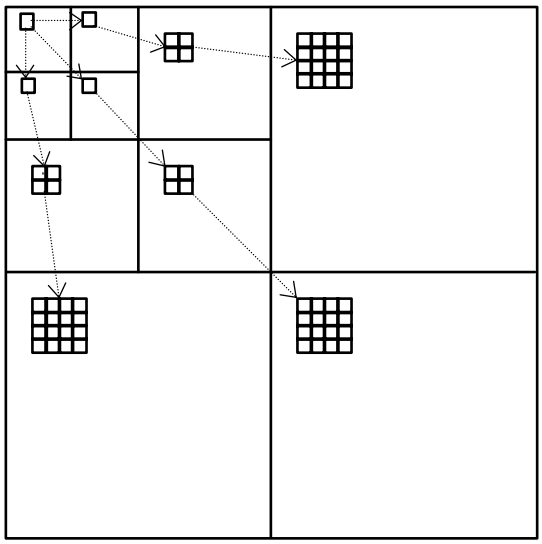


Figure 7-58 — Parent-child relationship of wavelet coefficients

In transform-based coding, it is typically true that a large percentage of the transform coefficients are quantised to zero. A substantial number of bits must be spent either encoding these zero-valued quantised coefficients, or else encoding the location of the non-zero-valued quantised coefficients. ZeroTree Coding uses a data structure called a *zerotree*, built on the parent-child relationships described above, and used for encoding the location of non-zero quantised wavelet coefficients. The zerotree structure takes advantage of the principle that if a wavelet coefficient at a coarse scale is "insignificant" (quantised to zero), then all wavelet coefficients of the same orientation at the same spatial location at finer wavelet scales are also likely to be "insignificant". Zerotrees exist at any tree node where the coefficient is zero and all its descendents are also zero.

The wavelet trees are efficiently represented and coded by scanning each tree from the root in the 3 lowest AC bands through the children, and assigning one of four symbols to each node encountered: *zerotree root (ZTR), value zerotree root (VZTR)*, *isolated zero (IZ)* or *value (VAL)*. A *ZTR* denotes a coefficient that is the root of a

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

zerotree. Zerotrees do not need to be scanned further because it is known that all coefficients in such a tree have amplitude zero. A *VZTR* is a node where the coefficient has a nonzero amplitude, and all four children are zerotree roots. The scan of this tree can stop at this symbol. An *IZ* identifies a coefficient with amplitude zero, but also with some nonzero descendant. A *VAL* symbol identifies a coefficient with amplitude nonzero, and with some nonzero descendant. The symbols and quantised coefficients are losslessly encoded using an adaptive arithmetic coder. Table 7-19 shows the mapping of indices of the arithmetic decoding model into the zerotree symbols:

Table 7-19 — The indexing of zerotree symbols

index	Symbol
0	IZ
1	VAL
2	ZTR
3	VZTR

In order to achieve a wide range of scalability levels efficiently as needed by different applications, three different zerotree scanning and associated inverse quantisation methods are employed. The encoding mode is speficied in bitstream with quantisation_type field as one of 1) single_quant, 2) multi_quant or 3) bilevel_quant:

Table 7-20 — The quantisation types

code	quantisation_type
01	single_quant
10	multi _quant
11	bilevel_quant

In single_quant mode, the bitstream contains only one zero-tree map for the wavelet coefficients. After arithmetic decoding, the inverse quantisation is applied to obtain the reconstructed wavelet coefficients and at the end, the inverse wavelet transform is applied to those coefficients.

In multi_quant mode, a multiscale zerotree decoding scheme is employed. Figure 7-59 shows the concept of this technique.

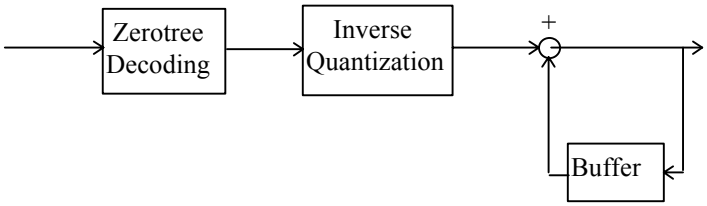


Figure 7-59 — Multiscale zerotree decoding

The wavelet coefficients of the first spatial (and/or SNR) layer are read from the bitstream and decoded using the arithmetic decoder. Zerotree scanning is used for decoding the significant maps and quantised coefficients and locating them in their corresponding positions in trees. These values are saved in the buffer to be used for quantisation refinement at the next scalability layer. Then, an inverse quantisation is applied to these indices to obtain the quantised wavelet coefficients. An inverse wavelet transform can also be applied to these coefficients to obtain the first decoded image. The above procedure is applied for the next spatial/SNR layers.

The bilevel_quant mode enables fine granular SNR scalability by encoding the wavelet coefficients in a bitplane-by-bitplane fashion. This mode uses the same zerotree symbols as the multi_quant mode. In this mode, a zero-tree map is decoded for each bitplane, indicating which wavelet coefficients are nonzero relative to that bitplane. The inverse quantisation is also performed bitplane by bitplane. After the zero-tree map, additional bits are decoded to refine the accuracy of the previously decoded coefficients.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.10.2.1 Zerotree Scanning

In all the three quantisation modes, the wavelet coefficients are scanned either in the tree-depth fashion or in the band-by-band fashion. In the tree-depth scanning order all coefficients of each tree are decoded before starting decoding of the next tree. In the band-by-band scanning order, all coefficients are decoded from the lowest to the highest frequency subbands.

Figure 7-60 shows the scanning order for a 16x16 image, with 3 levels of decomposition. In this figure, the indices 0,1,2,3 represent the DC band coefficients which are decoded separately. The remaining coefficients are decoded in the order shown in this figure. As an example, indices 4,5,..., 24 represent one tree. At first, coefficients in this tree are decoded starting from index 4 and ending at index 24. Then, the coefficients in the second tree are decoded starting from index 25 and ending at 45. The third tree is decoded starting from index 46 and ending at index 66 and so on.

0	1	4	67	5	6	68	69	9	10	13	14	72	73	76	77
2	3	130	193	7	8	70	71	11	12	15	16	74	75	78	79
25	88	46	109	131	132	194	195	17	18	21	22	80	81	84	85
151	214	172	235	133	134	196	197	19	20	23	24	82	83	86	87
26	27	89	90	47	48	110	111	135	136	139	140	198	199	202	203
28	29	91	92	49	50	112	113	137	138	141	142	200	201	204	205
152	153	215	216	173	174	236	237	143	144	147	148	206	207	210	211
154	155	217	218	175	176	238	239	145	146	149	150	208	209	212	213
30	31	34	35	93	94	97	98	51	52	55	56	114	115	118	119
32	33	36	37	95	96	99	100	53	54	57	58	116	117	120	121
38	39	42	43	101	102	105	106	59	60	63	64	122	123	126	127
40	41	44	45	103	104	107	108	61	62	65	66	124	125	128	129
156	157	160	161	219	220	223	224	177	178	181	182	240	241	244	245
158	159	162	163	221	222	225	226	179	180	183	184	242	243	246	247
164	165	168	169	227	228	231	232	185	186	189	190	248	249	252	253
166	167	170	171	229	230	233	234	187	188	191	192	250	251	254	255

Figure 7-60 — Tree depth scanning order of a wavelet block in the all three modes

Figure 7-61 shows that the wavelet coefficients are scanned in the subband by subband fashion, from the lowest to the highest frequency subbands. Figure 7-62 shows an example of decoding order for a 16x16 image with 3 levels of decomposition for the subband by subband scanning. The DC band is located at upper left corner (with indices 0, 1,2, 3) and is decoded separately as described in DC band decoding. The remaining coefficients are decoded in the order which is shown in the figure, starting from index 4 and ending at index 255. In multi_quant mode, at first scalability layer, the zerotree symbols and the corresponding values are decoded for the wavelet coefficients of that scalability layer. For the next scalability layers, the zerotree map is updated along with the corresponding value refinements. In each scalability layer, a new zerotree symbol is decoded for a coefficient only if it was decoded as ZTR or IZ in previous scalability layer or it is currently in SKIP mode. A node is said to be in SKIP mode when the number of quantisation refinement levels for the current scalability layer is one. The detailed description of the refinement of quantisation level is found in subclause 7.10.3. If the coefficient was decoded as VAL in previous layer and it is not currently in SKIP mode, a VAL symbol is also assigned to it at the current layer and only its refinement value is decoded from bitstream.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

0	1	4	7	16	17	28	29	64	65	68	69	112	113	116	117
2	3	10	13	18	19	30	31	66	67	70	71	114	115	118	119
5	8	6	9	40	41	52	53	72	73	76	77	120	121	124	125
11	14	12	15	42	43	54	55	74	75	78	79	122	123	126	127
20	21	32	33	24	25	36	37	160	161	164	165	208	209	212	213
22	23	34	35	26	27	38	39	162	163	166	167	210	211	214	215
44	45	56	57	48	49	60	61	168	169	172	173	216	217	220	221
46	47	58	59	50	51	62	63	170	171	174	175	218	219	222	223
80	81	84	85	128	129	132	133	96	97	100	101	144	145	148	149
82	83	86	87	130	131	134	135	98	99	102	103	146	147	150	151
88	89	92	93	136	137	140	141	104	105	108	109	152	153	156	157
90	91	94	95	138	139	142	143	106	107	110	111	154	155	158	159
176	177	180	181	224	225	228	229	192	193	196	197	240	241	244	245
178	179	182	183	226	227	230	231	194	195	198	199	242	243	246	247
184	185	188	189	232	233	236	237	200	201	204	205	248	249	252	253
186	187	190	191	234	235	238	239	202	203	206	207	250	251	254	255

Figure 7-61 — The band-by-band scanning order for all three modes

In bilevel_quant mode, the band by band scanning is also employed, similar to the multi_quant mode. When bi-level quantisation is applied, the coefficients that are already found significant are replaced with zero symbols for the purpose of zero-tree forming in later scans.

7.10.2.2 Entropy Decoding

The zero-tree (or type) symbols, quantised coefficient values (magnitude and sign), and residual values (for the multi quant mode) are all decoded using an adaptive arithmetic decoder with a given symbol alphabet. The arithmetic decoder adaptively tracks the statistics of the zerotree symbols and decoded values. For both the single quant and multi quant modes the arithmetic decoder is initialized at the beginning of each color loop for band-by-band scanning and at the beginning of the tree-block loop for tree-depth scanning. Therefore, the decoder is initialized three times in each band for band-by-band scanning and the decoder is initialized once before going to the tree block-loop for tree-depth scanning. In order to avoid start code emulation, the arithmetic encoder always starts with stuffing one bit '1' at the beginning of the entropy encoding. It also stuffs one bit '1' immediately after it encodes every 22 successive '0's. It stuffs one bit '1' to the end of bitstream in the case in which the last output bit of arithmetic encoder is '0'. Thus, the arithmetic decoder reads and discards one bit before starts entropy decoding. During the decoding, it also reads and discards one bit after receiving every 22 successive '0's. The arithmetic decoder reads one bit and discards it if the last input bit to the arithmetic decoder is '0'. To reduce the number of bitplanes in encoding the magnitude of quantised coefficient values, the encoder subtracts the magnitude by one first. Thus the decoder adds one back after decoding the magnitude of quantised coefficient values.

The context models used for SQ and MQ are identical to the ones used in BQ mode.

For both scanning orders in the single quant and multi_quant modes separate probability models are kept for each color and wavelet decomposition layer for the type and sign symbols while separate probability models are kept for each color, wavelet decomposition layer, *and* bitplane for the magnitude and residual symbols. All the models are initialized with a uniform probability distribution.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The models and symbol sets for the non-zerotree type quantities to be decoded are as follows:

Model	Possible Values
Sign	POSITIVE (0), NEGATIVE (1)
Magnitude	0, 1
Residual	0, 1

The possible values of the magnitudes and residuals are only 0 or 1 because each bitplane is being decoded separately. The non-residual values are decoded in two steps. First, the absolute value is decoded in a bitplane fashion using the *magnitude* probability models, then its sign is decoded.

For the decoding of the type symbols different probability models are kept for the leaf and non-leaf coefficients. For the multi quant mode, context modeling, based on the zerotree type of the coefficient in the previous scalability layer, is used. The different zerotree type models and their possible values are as follows:

Context and Leaf/Non-Leaf	Possible Values
INIT	ZTR (2), IZ (0), VZTR (3), VAL (1)
ZTR	ZTR (2), IZ (0), VZTR (3), VAL (1)
ZTR DESCENDENT	ZTR (2), IZ (0), VZTR (3), VAL (1)
IZ	IZ (0), VAL (1)
LEAF INIT	ZTR (0), VZTR (1)
LEAF ZTR	ZTR (0), VZTR (1)
LEAF ZTR DESCENDENT	ZTR (0), VZTR (1)

For the single quant mode only the INIT and LEAF INIT models are used. For the multi quant mode for the first scalability layer only the INIT and LEAF INIT models are used. Subsequent scalability layers in the multi quant mode use the context associated with the type. If a new spatial layer is added then the contexts of all previous leaf band coefficients are switched to the corresponding non-leaf contexts. The coefficients in the newly added bands use the LEAF INIT context. The residual models are used to decode the coefficient refinements if in the previous layer, a *VZTR or VAL* symbol was assigned. If a node is currently not in SKIP mode (meaning that no refinement is being done for the coefficient – see subclause 7.10.3 on inverse quantisation for details) only the magnitude of the refinements are decoded as these values are always zero or positive integers.

If a node is in SKIP mode, then its new zerotree symbol is decoded from bitstream, but no value is decoded for the node and its value in the current scalability layer is assumed to be zero.

States in Previous Bitplane	Possibilities in current bitplane
ZTR	ZTR, VZTR, IZ, VAL
VZTR	SKIP
IZ	IZ, VAL
VAL	SKIP
DZTR	ZTR, VTRZ, IZ, VAL

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For the bi-level quantisation mode, the zero-tree map is decoded for each bitplane, indicating which wavelet coefficients are zeros relative to the current quantisation step size. Different probability models for the arithmetic decoder are used and updated according to the local contexts. For decoding the zerotree symbols, five context models are used, which are dependent on the status of the current wavelet coefficients in the zerotree formed in the previous bitplane decoding. Specifically, the five models correspond to the following contexts of the current wavelet coefficient:

· IZ: the previous zerotree symbol is Isolated Zero.

· VAL: the previous zerotree symbol is Value.

· ZTR: the previous zerotree symbol is Zerotree Root.

· VZTR: the previous zerotree symbol is valued zerotree root.

· DZTR: in previous bitplane, the current coefficient is a descendant of a zerotree root

The additional symbol DZTR is used for switching the models only, where DZTR refers to the descendant of a ZTR symbol. The context symbols DZTR can be inferred from the decoding process and are not included in the bitstream. They are used for switching the models only. At the beginning of the decoding the first bitplane, the contexts of the coefficients are initialized to be DZ. For the highest subband, only IZ and VAL are possible (no ZTR and VZTR are possible). Therefore, we initialize the arithmetic model for the last band differently (with zero probablility for ZTR and VZTR symbols).

For decoding the sign information, another context model (the sign model) is used and updated. For decoding the refinement bits, another statistical model (the refinement model) is used

Each decomposition levels have their own separate arithmetic models. Therefore, the above decoding process applies to each decomposition levels. All models are initialized at the beginning of coding each bitplane.

After the zero-tree map, additional bits are received to refine the accuracy of the coefficients that are already marked significant by previously received information at the decoder. For each significant coefficient, the 1-bit bi-level quantised refinement values are entropy coded using the arithmetic coder.

7.10.3 Inverse Quantisation

Different quantisation step sizes (one for each color component) are specified for each level of scalability. The quantiser of the DC band is a uniform mid-step quantiser with a dead zone equal to the quantisation step size. The quantisation index is a signed integer number and the quantisation reconstructed value is obtained using the following equation:

$$V = id * Qdc,$$

where V is the reconstructed value, *id* is the decoded index and Qdc is the quantisation step size.

All the quantisers of the higher bands (in all quantisation modes) are uniform mid-step quantiser with a dead zone 2 times the quantisation step size. For the single quantisation mode, the quantisation index is an signed integer. The reconstructed value is obtained using the following algorithm:

```
if (id == 0)
    V =0;
else if ( id > 0 )
    V = id*Q+Q/2;
else
    V = id*Q-Q/2;
```

where V is the reconstructed value, id is the decoded index and Q is the quantisation step size.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the multi- quantisation mode each SNR layer within each spatial layer has an associated quantisation step-size value (Q value). These different Q Values are used for SNR scalability. A lower Q Value will result in a more accurate reconstruction.

If a coefficient is in a given spatial layer it is also in all higher numbered spatial layers. SNR scalability may be continued on these coefficients in the higher numbered spatial layers in the same way as is done in the spatial layer the coefficient first arises in. Thus, we can think of all the coefficients which first arise in a particular spatial layer as having a corresponding sequence of Q Values (call it a Q Sequence). The Q Sequence is made up of the quantisation values for all SNR layers in the spatial layer the coefficient first arises in plus the quantisation values in all SNR layers in all higher spatial layers. The order is from lower to higher numbered spatial layers and from lower to higher numbered SNR layers within each spatial layer.

EXAMPLE

Let the quantisation value of the *i*-th spatial layer and the *j*-th SNR layer be denoted by *Q(i,j)*. Assume we have the following scenario:

Spatial	SNR Layer		
Layer	0	1	2 .
0	*Q(0,0)*	*Q(0,1)*	*Q(0,2)*
1	*Q(1,0)*	*Q(1,1)*	*Q(1,2)*
2	*Q(2,0)*	*Q(2,1)*	*Q(2,2)*

The Q Sequence which will be used to quantise all coefficients which first arise in spatial layer 0 is:

<Q(0,0) Q(0,1) Q(0,2) Q(1,0) Q(1,1) Q(1,2) Q(2,0) Q(2,1) Q(2,2)>

while the sequence for all coefficients first arising in spatial layer 1 is:

<Q(1,0) Q(1,1) Q(1,2) Q(2,0) Q(2,1) Q(2,2)>

and the sequence for all coefficients first arising in spatial layer 2:

<Q(2,0) Q(2,1) Q(2,2)>.

As in the single- quantisation case we would like to have a uniform quantiser for all layers. Due to the manner in which the Q Values are used to achieve scalability (described below), in order to have a (approximately) uniform quantiser at each layer, we may have to revise the Q Values extracted from the bitstream before reconstruction. This revision is necessary if the Q Values within each Q Sequence are not integer multiples of one another or if Q Value is greater than a Q Value occurring earlier in the Q Sequence.

EXAMPLES

Q Sequences needing no revision: <*24 8 2*> and <*81 81 27*>.

Q Sequences needing revision: <*31 9 2*> (non-integer multiples) and <*81 162 4*> (increasing Q Value).

If a coefficient's quantisation indices have been zero for all previous scalability layers (spatial and SNR) or if it is the first scalability layer, then the reconstruction is the similar to the single-quantisation mode described above. The difference is in that the refined Q Values may be used instead of the ones extracted from the bitstream. The refinement process is described below in steps 1 and 2. If there has been a non-zero quantisation index in a previous scalability layer then the quantisation index specifies a refinement of the previous quantisation. The indices are then called residuals. For every coefficient and scalability layer we know (1) the quantisation interval where the coefficient occurred in the last scalability layer (both size and location), (2) the spatial layer the coefficient first arose in (and thus, which Q Sequence to use), (3) the current Q Value and the previous Q Value in the Q Sequence, and (4) the refinment (if any) of the previous Q Value.

The reconstruction of the residual is calculated in the following five steps.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 1: Calculation of the Number of Refinement Intervals

The quantisation interval which was indexed in the previous layer is to be partitioned into disjoint intervals. The number of these "refinement" intervals is calculated based solely on the current Q Value (call it *curQ*) and the previous Q Value (call it *prevQ*). Note that *prevQ* may have been revised as mentioned above. Letting *m* be the number of refinement intervals we calculate

m = ROUND(prevQ÷curQ)
where ROUND(x) = MAX(nearest integer of x, 1)).

If *m = 1*, no refinement is needed and no value will have been sent. If, at a certain scalability layer, a node has *m=1* then it is said to be in SKIP mode. Thus, steps 2, 3, and 4 need not be performed for the coefficient.

Step 2: Calculation of the Maximum Refinement Interval Size

Using the number of refinement intervals, the current Q Value, *curQ*, is revised (if necessary).

curQ = CEIL(prevQ ÷ m)
where CEIL rounds up to the nearest integer.

curQ represents the maximum size of the intervals in the partition. Since *prevQ* is the previous layer's *curQ* (see step 5), we see that *prevQ* represents the maximum size of the intervals in the partition used in the previous scalability layer.

Step 3: Construction of Refinement Partition

Using the values *m* and *curQ* calculated above and the size of quantisation interval where the coefficient occurred in the last scalability layer, we form the refinement partition.

The previous layer's quantisation interval is partitioned into *m* intervals which are of size *curQ* or *curQ-1*. The residual will be one of the values *{0, 1, ..., m-1}* which represent an index into this partition. A lower number index corresponds to an interval in the partition covering lower magnitude values. If the partition is made up of different size intervals (*curQ* and *curQ-1*) then the *curQ* size intervals correspond to the lower indices. Some combination of m *curQ* and *curQ-1* interval sizes are sufficient to cover the previous quantisation interval. From step 2 we know that the previous quantisation interval is of size *prevQ* or *prevQ-1*.

Step 4: Calculation of Reconstructed Value

The interval in the partition indexed by the residual is mapped to the reconstruction value. The reconstruction is just the middle point of the interval in the partition that the residual indexes. That is, if *PartStart* is the start of the interval in the partition which is indexed by the residual, *PartStartSize* is the size of the interval, *sign* is the corresponding sign (known from prior scalability layers), and // is integer division then the reconstructed value is:

PartStart + sign*(PartStartSize-1)//2

Step 5: Assignment of Maximum Size

If there is another scalability layer then *prevQ* is assigned the value of *curQ*.

Note that since steps 1, 2, and 5 depend entirely on the Q Values found in the Q Sequences they only need to be done once in each scalability layer for each Q Sequence being used in the current spatial layer.

FOUR EXAMPLES

In the examples:

1. let the Q Values be Q1 , Q2 , and Q3,
2. let two sample coefficients to be quantised be *C1* and *C2,*
3. let *Cq1* and *Cq2* denote the corresponding quantised coefficients or residuals, and
4. let *iC1* and *iC2* denote the corresponding reconstructed coefficient values.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1. Q Values not in need of revision

Q1 = 24 , Q2 = 8, Q3 = 2,
C1 = 16, and C2 = 28.

At first scalability layer we have

Cq1 = C1/Q1 = 0
iCq1 = 0
Cq2 = C2/Q1 = 1
iCq2 = 35

At second scalability layer we have

prevQ = Q1 = 24
curQ = Q2 = 8
m = ROUND(prevQ÷curQ) = ROUND(24÷8) = 3
curQ = CEIL(prevQ÷m) = CEIL(24÷3) = 8
partition sizes = {8, 8, 8}
Cq1 = C1/curQ = 2
iCq1 = 19
Cq2 = 0 (residual)
iCq2 = 27

At third scalability layer we have

prevQ = curQ = 8
curQ = Q3 = 2
m = ROUND(prevQ÷curQ) = ROUND(8÷2) = 4
curQ = CEIL(prevQ÷m) = CEIL(8÷2) = 4
partition sizes = {2, 2, 2, 2}
Cq1 = 0 (residual)
iCq1 = 16
Cq2 = 2 (residual)
iCq2 = 28

2. Q Values not in need of revision

Q1 = 81, Q2 = 81, Q3 = 27,
C1 = 115 , and C2 = 28.

At first scalability layer we have

Cq1 = C1/Q1 = 1
iCq1 = 121
Cq2 = C2/Q1 = 0
iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 81
curQ = Q2 = 81
m = ROUND(prevQ÷curQ) =ROUND(81÷81) = 1
curQ = CEIL(prevQ÷m) = CEIL(81÷1) = 81
partition sizes = {81} (no refinement needed)
Cq1 = 0 (residual)
iCq1 = 121
Cq2 = C2/curQ = 0
iCq2 = 0

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

At third scalability layer we have

prevQ = curQ = 81
curQ = Q3 = 27
m = ROUND(prevQ÷curQ) = ROUND(81÷27) = 3
curQ = CEIL(prevQ÷m) = CEIL(81÷3) = 27
partition sizes = {27, 27, 27}
Cq1 = 1 (residual)
iCq1 = 121
Cq2 = C2/curQ = 1
iCq2 = 40

3. Q Values in need of revision

Q1 = 31 , Q2 = 9, Q3 = 2,
C1 = 115 , and C2 = 5.

At first scalability layer we have

Cq1 = C1/Q1 = 3
iCq1 = 108
Cq2 = C2/Q1 = 0
iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 31
curQ = Q2 = 9
m = ROUND(prevQ÷curQ) = ROUND(31÷9) = 3
curQ = CEIL(prevQ÷m) = CEIL(31÷3) = 11
partition sizes = {11, 10, 10}
Cq1 = 2 (residual)
iCq1 = 118
Cq2 = C2/curQ = 0
iCq2 = 0

At third scalability layer we have

prevQ = curQ = 11
curQ = Q3 = 2
m = ROUND(prevQ÷curQ) = ROUND(11÷2) = 6
curQ = CEIL(prevQ÷m) = CEIL(11÷6) = 2
partition sizes = {2, 2, 2, 2, 2,1} if value occurs in level with size 11
partition sizes = {2, 2, 2, 2, 1, 1} if value occurs in level with size 10
Cq1 = 0 (residual)
iCq1 = 114
Cq2 = C2/curQ = 2
iCq2 = 4

4. Q Values in need of revision

Q1 = 81, Q2 = 162, Q3 = 4,
C1 = 115 , and C2 = 5.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

At first scalability layer we have

Cq1 = C1/Q1 = 1
iCq1 = 121
Cq2 = C2/Q1 = 0
iCq2 = 0

At second scalability layer we have

prevQ = Q1 = 81
curQ = Q2 = 162
m = ROUND(prevQ÷curQ) = ROUND(81÷162) = 1
curQ = CEIL(prevQ÷m) = CEIL(81÷1) = 81
partition sizes = {81} (no refinement needed)
Cq1 = 0 (not used)
iCq1 = 121
Cq2 = C2/curQ = 0 (not used)
iCq2 = 0

At third scalability layer we have

prevQ = curQ = 81
curQ = Q3 = 4
m = ROUND(prevQ÷curQ) = ROUND(81÷4) = 20
curQ = CEIL(prevQ÷m) = CEIL(81÷20) = 5
partition sizes = {5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4}
Cq1 = 8 (residual)
iCq1 = 121
Cq2 = C2/curQ = 1
iCq2 = 6

It is important to note that coefficients which first arose in different spatial layers may use different *prevQ* and *curQ* values. They are being quantised from different lists. That is, they have different corresponding Q Sequences.

7.10.3.1 Shape adaptive zerotree decoding

When the texture_object_layer_shape is not rectangular or **texture_object_layer_width**!= integer*$2^{\textbf{wavelet_decomposition_level}}$ or **texture_object_layer_height**!= integer*$2^{\textbf{wavelet_decomposition_level}}$, the inverse shape adaptve wavelet transform is chosen to reconstructed the arbitrary-shaped image object or rectangular texture object. Decoding shape adaptive wavelet coefficients is the same as decoding regular wavelet coefficients except keep track of the locations of where to put the decoded wavelet coefficients according to the shape information. or a generated mask. The mask is generated with **texture_object_layer_width*texture_object_layer_height** pixels of value 255 at the upper-left corner of a frame of size w*$2^{\text{wavelet_decomposition_level}}$ * h*$2^{\text{wavelet_decomposition_level}}$ and the rest of the pixels being value 0, where w is the smallest integer that makes w*$2^{\text{wavelet_decomposition_level}}$ > **texture_object_layer_width** and h is the smallest integer that makes h*$2^{\text{wavelet_decomposition_level}}$ > **texture_object_layer_height**. Similar to decoding of regular wavelet coefficients, the decoded zerotree symbols at a lower subband are used to determine whether decoding is needed at higher subbands. The difference is now that some zerotree nodes correspond to the pixel locations outside the shape boundary and no bits are to be decoded for these out_nodes. Root layer is defined as the lowest three AC subbands, leaf layer is defined as the highest three AC subbands. For decomposition level of one, the overlapped root layer and leaf laver shall be treated as leaf layer.

7.10.3.1.1 DC layer

The DC coefficient decoding is the same as that for rectangular image except the following,

1. Only those DC coefficients inside the shape boundary in the DC layer shall be traversed and decoded and DC coefficients outside the shape boundary may be set to zeros.
2. For the inverse DC prediction in the DC layer, if a reference coefficient (Figure 7-57) in the prediction context is outside the shape boundary, zero shall be used to form the prediction syntax.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

7.10.3.1.2 Root layer

At the root layer (the lowest 3 AC bands), the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;
- the four children nodes of this node are marked “to_be_decoded” (TBD);

otherwise,

- a zerotree symbol is decoded for this node using an adaptive arithmetic decoder.

If the decoded symbol for the node is either isolated_zero (IZ) or value (VAL),

- the four children nodes of this node are marked TBD;

otherwise,

- the symbol is either zerotree_root (ZTR) or valued_zerotree_root (VZTR) and the four children nodes of this node are marked “no_code” (NC).

If the symbol is VAL or VZTR,

- a non-zero wavelet coefficient is decoded for this node by root model;

otherwise,

- the symbol is either IZ or ZTR and the wavelet coefficient is set to zero for this node.

7.10.3.1.3 Between root and leaf layer

At any layer between the root layer and the leaf layer, the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;
- the four children nodes of this node are marked as either TBD or NC depending on whether this node itself is marked TBD or NC respectively;

otherwise, if it is marked NC,

- no bits are decoded for this node;
- the wavelet coefficient is set to zero for this node;
- the four children nodes are marked NC;

otherwise,

- a zerotree symbol is decoded for this node using an adaptive arithmetic decoder.

If the decoded symbol for the node is either isolated_zero (IZ) or value (VAL),

- the four children nodes of this node are marked TBD;

otherwise,

- the symbol is either zerotree_root (ZTR) or valued_zerotree_root (VZTR) and the four nodes of this node are marked “no_code” (NC).

If the symbol is VAL or VZTR,

- a non-zero wavelet coefficient is decoded for this node by valnz model;

otherwise,

- the symbol is either IZ or ZTR and the wavelet coefficient is set to zero for this node.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.10.3.1.4 Leaf layer

At the leaf layer, the shape information is examined for every node to determine whether a node is an out_node.

If it is an out_node,

- no bits are decoded for this node;

otherwise, if it is marked NC,

- no bits are decoded for this node;
- the wavelet coefficient is set to zero for this node;

otherwise,

- * a wavelet coefficient is decoded for this node by valz adaptive arithmetic model;

7.10.3.2 Shape decomposition

The shape information for both shape adaptive zerotree decoding and the inverse shape adaptive wavelet transform is obtained by decomposing the reconstructed shape from the shape decoder. Assuming binary shape with 0 or 1 indicating a pixel being outside or inside the arbitrarily shaped object, the shape decomposition procedure can be described as follows:

- For each horizontal line, collect all even-indexed shape pixels together as the shape information for the horizontal low-pass band and collect all odd-indexed shape pixels together as the shape information for the horizontal high-pass band, except for the special case where the number of consecutive 1's is one.
- For an isolated 1 in a horizontal line, whether at an even-indexed location or at an odd-indexed location, it is always put together with the shape pixels for the low-pass band and a 0 is put at the corresponding position together with the shape pixels for the high-pass band.
- Perform the above operations for each vertical line after finishing all horizontal lines.
- Use the above operations to decompose the shape pixels for the horizontal and vertical low-pass band further until the number of decomposition levels is reached.

7.10.4 Still Texture Error Resilience

In error resilience mode, start code will always be used. The decoder is expecting packetized bitstream. The first packet is the image header information, then the shape coding information (for shape coding combined with error resilience, each scalable shape layer will be one packet starting with the texture_shape_layer_start_code), followed by the packets for DC and AC bands. Each packet start with texture marker (texture_marker = '00000000000000001') or a startcode (if the packet is the first in the layer), followed by the number of the first and last texture units encoded in the texture packet, named TU_first and TU_last, and by one bit header_extension_code, which is optionally followed by header information.

The TU_first and TU_last provide absolute reference for the texture units in the packet. This parameter specifies the texture unit number within a texture object. These two parameters are decoded by the function get_param(). The labeling of the texture units starts from 0 in DC band and following the same decoding order as in the non-error resilient case. In AC band, the TU number starts from 60 (> maximum number of bitplanes in DC band).

The header_extention_code bit has value 1 to indicate that additional header information is sent in the packet, zero otherwise, depending on error resilience needs. The information following the header_extention_code repeats the header information contained in the TextureObjectLayer().

Within each packet, segment_markers (a ZTR symbol coded by arithmetic coding) are expected when large segment is encountered. A segment is formed at the end of the currently decoded *sub_unit* (a part of a texture unit, to be defined in the following) whenever the total number of bits from the end of the last segment or from the beginning of the packet exceeds a threshold target_segment_length. The segment marker is decoded using the function decode_segment_marker().

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.10.4.1 Decoder Constraints

In order to support error resilience, the following constrains are used in the TOL.

To prevent error propagation, DPCM prediction in DC part is disabled.

To assure that the symbol identifying the resynch marker is unique, we impose the following constraint to the arithmetic decoder (AD): when the AD decodes the 000000000000000 (15 zero bits) symbol while decoding the wavelet coefficients, it expect the above bits to be followed by one '1' bit.

The AD has to reset all the models used in the adaptation process when starting a new texture packet. This assures that there are no dependencies across the texture packets. The models will be reset to uniform distribution.

7.10.4.2 Texture packet structure

When the TOL is operating in error resilient mode, the basic unit of a texture packet is the stream generated by encoding one texture unit. In the TOL error resilient case we define four types of texture units. Each texture unit lies entirely within one scalability layer.

7.10.4.2.1 DC band

In DC band, the texture unit is defined as a bitplane of all DC coefficients, starting from the MSB. The sub-unit is defined as one row of pixels in a texture unit.

7.10.4.2.2 Tree-depth scanning

In the tree-depth scanning mode the AC texture unit consists of the stream generated when encoding one tree structure, as specified in Figure 7-62. Black blocks represent possible resynch marker positions. The total number of texture block is a function of the image size and of the number of levels in the wavelet decomposition. The texture blocks are formed following the scanning used to encode the coefficient. This is done to avoid degradation in coding efficiency performance.

The sub_unit are defined as follows:

All pixels in levels 0 and 1 within a tree.

The $2^l \times 2^l$ block within a tree for levels l = 2, 3.

The 16x16 block within a tree for levels l >3.

An example is shown in Figure 7-63.

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

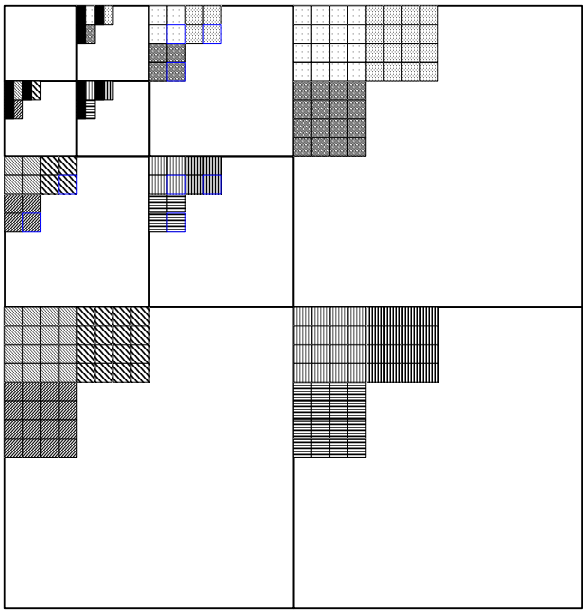


Figure 7-62 — Texture units for the single_quant mode.
All coefficients with the same shade belong into the same texture unit

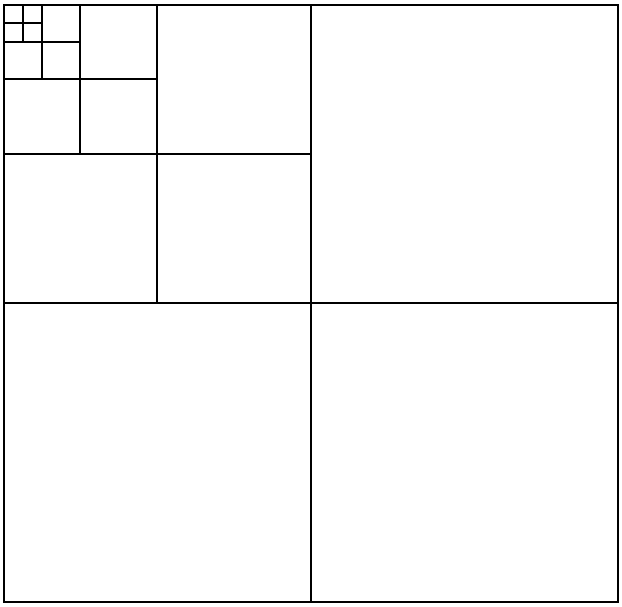


Figure 7-63 — Sub-unit formation in a wavelet decomposed image.
Each type of shade represents a sub-unit.

7.10.4.2.3 Subband-by-subband scanning

In the subband-by-subband scanning mode, the AC texture unit consists of the bit stream generated when encoding a slice of subband coefficients from all three subbands in the same wavelet decomposition layer, as specified in Figure 7-64. The subbands are scanned using the same scanning order used to encode the coefficients.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The sub_unit is defined in as follows:

1) All pixels at level 0 within a TU

2) The $2^l \times 2^l$ blocks within a TU for levels l = 1, 2, 3.

3) The 16x16 blocks within a TU for levels l >3.

Please note that this approach does not break the scalability, as the order of the bits generated by the encoder is preserved. Also, note that in the subband-by-subband scanning mode there is a strong dependency across subbands. The dependency consists of the having a part of the zero-tree significance map coded together with the subband coefficients. Because of this, losing a part of the subband means to lose part of the coefficients in the lower subbands.

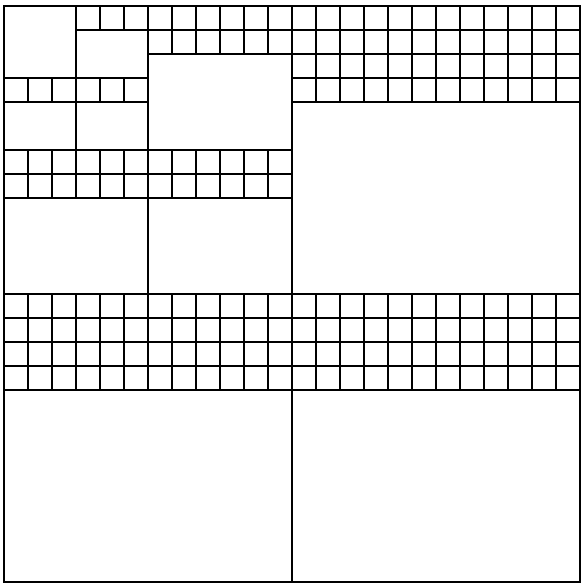


Figure 7-64 — Texture units for the band_by_band mode.
All coefficients with the same shade belong to the same texture unit.

7.10.5 Wavelet Tiling

The block diagram of the decoder supporting the Wavelet Tiling is shown in Figure 7-65.

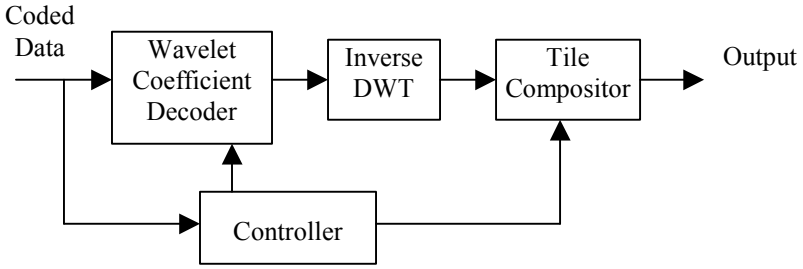


Figure 7-65 — Block diagram of the decoder with tiling option

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The module ‘Controller’ extracts the control information from coded data. It signals the ‘Wavelet coefficient Decoder’ module to decode a target tile based on the information. The target tile is decided by a system layer request. ‘Controller’ also signals ‘Tile Compositor’ module in order to specify a position of a decoded tile described by ‘tile_id’.

The module of ‘Wavelet Coefficient Decoder’ is basically equivalent to the all modules without Inverse DWT appeared in Figure 7-56. ‘Inverse DWT’ is equivalent to the modules appeared in Figure 7-56. ‘Tile Compositor’ composes the decoded tiles based on the signal from ‘Controller’ and output single Still Texture Object.

7.10.5.1 Structure of Tile

The structure of Still Texture Object with wavelet tiling is shown in Figure 7-66. This picture describes the case when texture_object_layer_shape==’00’. When texture_object_layer_shape==’01’, object_width and object_height are used instead of texture_object_layer_width and texture_object_layer_height.

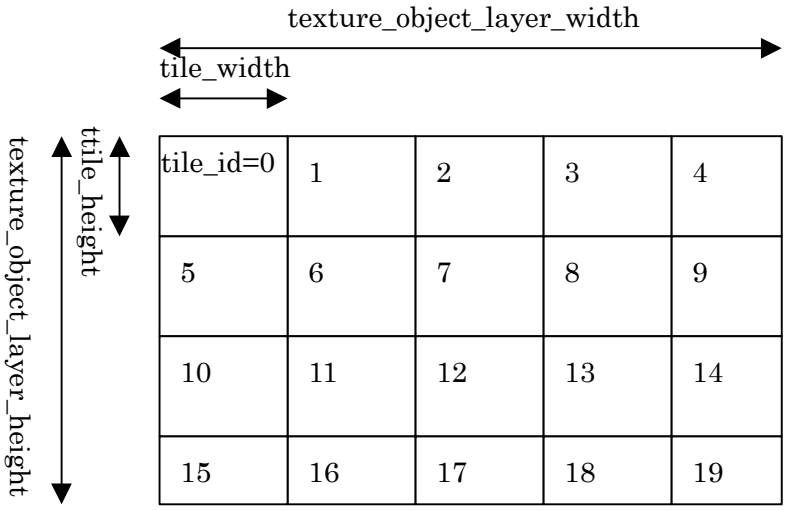


Figure 7-66 — Structure of tile

The relative position of each tile is identified by tile_id.

The number of tiles in row is computed by CEIL(texture_object_layer_width÷tile_width) and the number of tiles in column is computed by CEIL(texture_object_layer_height÷tile_height) when texture_object_layer_shape==’00’. Otherwise, object_width and object_height are used instead of texture_object_layer_width and texture_object_layer_height. Where CEIL() rounds up to the nearest integer.

7.10.5.2 The tile size of last row and last column

With regard to the tile size of last row and last column, texture_object_layer_width % tile_width and texture_object_layer_height % tile_height are used when texture_object_layer_shape==’00’ if texture_object_layer_width and texture_object_layer_height are not a multiple of tile_width and tile_height. When texture_object_layer_shape==’01’, object_width and object_height are used instead of texture_object_layer_width and texture_object_layer_height.

7.10.5.3 Searching a start position of the tile

The starting position of the target tile in the bitstream is found by using a look-up table before starting to decode. This way is applicable in tiling_jump_table_enable=1. In case of tiling_jump_table_enable = 0, texture_tile_start_code is used to find the start point of bitstream for each tile. For the look-up table generation, a size of each tile is stored as ‘tile_size_low’ and ‘tile_size_high’ in a header field. The decoding start position; *position[i]* for the *i*-th tile can be calculated by

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$position[i] = \sum_{j=0}^{i-1} tile_size[j]$$

where the bitstream size of *i*-th tile is expressed by *tile_size[i]*. This mechanism enables to decode the wavelet coefficients of the tile at any place in the entire image.

7.10.6 Scalable binary shape object decoding

At first binary_arithmetic_decode() is called for decoding the base layer shape. For decoding the shape object of each layer in the enhancement layer, enh_binary_arithmetic_decode() is performed in the macroblock decoding manner. For decoding each shape block, SI_bab_type is decoded in first. SI_bab_type denotes the BAB type of SI(Scan Interleaving). And then decoding for BAB pixels based on CAE is performed as the BAB type of SI.

7.10.6.1 Decoding of base layer

It is identical as the video non-scalable shape coding. Refer to the shape decoding for base layer in the subclause 7.5 Shape Coding.

7.10.6.2 Decoding of enhancement layer

7.10.6.2.1 Block size control

The BAB size is controlled before decoding shape object of each enhancement layer. Decision of BAB size depends on the object size of each scalable layer. The resolutions of objects are divided into 3 classes. If the current coding layer is the *L*-th layer in the bitstream. the BAB size in each object resolution is obtained as follows:

```
layer_width = object_width >> (wavelet_decomposition_levels – L – 1);

layer_height = object_height >> (wavelet_decomposition_levels – L – 1);

if( layer_width > = 1024 || layer_height > = 1024)      bab_size = 64;

else if( layer_width > = 512 || layer_height > = 512)   bab_size = 32;

else                                                    bab_size = 16;
```

7.10.6.2.2 Border formation

When decoding a BAB, pixels from neighbouring BABs can be used to make up the context. A 2 pixel wide border about the current BAB is used as depicted in Figure 7-67. In the figure, the size of the current BAB can be changed as described in subclause 7.10.6.2.1.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

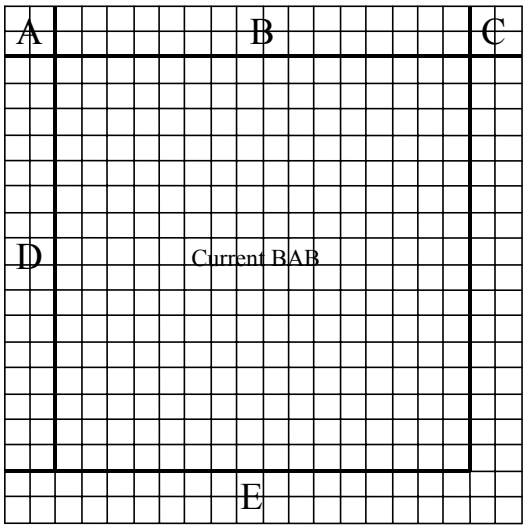


Figure 7-67 — Bordered BAB. A: TOP_LEFT_BORDER. B: TOP_BORDER. C: TOP_RIGHT_BORDER. D: LEFT_BORDER. E: BOTTOM_OR_RIGHT_BORDER

The border of the current BAB is partitioned into 5:

- TOP_LEFT_BORDER, which contains pixels from the BAB located to the upper-left of the current BAB and which consists of 2 lines of 2 pixels
- TOP_BORDER, which contains pixels from the BAB located above the current BAB and which consists of 2 lines of the number of bab_size pixels
- TOP_RIGHT_BORDER, which contains pixels from the BAB located to the upper-right of the current BAB and which consists of 2 lines of 2 pixels
- LEFT_BORDER, which contains pixels from the BAB located to the left of the current BAB and which consists of 2 columns of the number of bab_size pixels
- BOTTOM_OR_RIGHT_BORDER, which contains pixels from the collocated pixels in the lower layer and which consists of 2 lines in bottom and right of the current BAB. The pixel in (i,j) of the lower layer is repeatedly used for this border pixels in (2i,2j), (2i+1,2j), (2i,2j+1), and (2i+1,2j+1) of the current layer.

7.10.6.2.3 Decoding of BAB type of Scan Interleaving

For decoding a BAB of enhancement layer shape object, BAB type of SI is determined in advance. SI_bab_type denotes the two fields of BAB type of SI that are transitional BAB and exceptional BAB. Transitional BAB and exceptional BAB of SI are defined as follow.

transitional BAB: When the input BAB has no ESD and there exists no swapping pixel from wavelet decomposition, the input BAB is determined as transitional BAB. For decoding transitional BAB, only the TSD data of the BAB are decoded by BAC. All_0 and all_255 BAB are also included in this SI_bab_type. Since all_0 or all_255 BAB has no ESD or TSD, the BAB can be reconstructed by decoding only SI_bab_type.

exceptional BAB: When the input BAB has ESD or When there exist some swapping pixels from wavelet decomposition, the input BAB is determined as exceptional BAB. For decoding exceptional BAB, all of the pixels in the BAB are decoded by BAC.

The first decoded symbol of enh_binary_arithmetic_decode() is SI_bab_type. SI_bab_type is set to '0' for transitional BAB and is set to '1' for exceptional BAB. For decoding this flag by BAC, SI_bab_type_prob[] is used for the probability table of odd symmetry filter case, and sto_SI_bab_type_prob_even[] is used for the probability table of even symmetry filter case.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.10.6.2.4 Scan order decision

If SI_bab_type is '0' (transitional BAB) the scan order should be determined before performing BAC. The decision method is similar to the method in the subclause 7.5.4.11 except that it uses right and bottom neighboring pixels instead of left and upper neighboring pixels. The bottom and right borders of the collocated block of the lower layer are extended with its neighboring pixels like Figure 7-68. If an extended border is outside VOP, the pixel values of the extended border are set to zero as in the subclause 7.5.4.11.

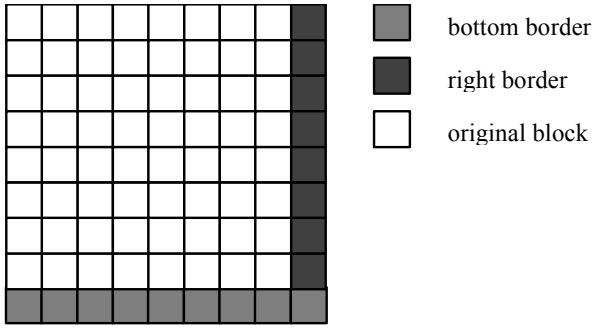


Figure 7-68 — Extension of border pixels

7.10.6.2.5 Decoding of transitional BAB

The decoding of transitional BAB is almost same with the scalable shape decoding of transitional BAB for P-VOP in the subclause 7.5.4 except that the decoding order of pixels inside the transitional BAB for still texture objects is different from the method in the subclause 7.5.4.8.

An example of decoding order of transitional BAB with size 16 x 16 (bab_size==16) is shown in Figure 7-69. In the figure, the "R"s indicate the pixels copied from collocated pixels in the reference layer. Apart from that, the actual decoding is based on the scan interleaving (SI) method as in the subclause 7.5.4.10. The binary arithmetic decoding and the usage of context values for arithmetic decoding are the same as those of the subclauses 7.5.4.13 and 7.5.4.14, respectively.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

R	0	R	8	R	16	R	24	R	32	R	40	R	48	R	56
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
R	1	R	9	R	17	R	25	R	33	R	41	R	49	R	57
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
R	2	R	10	R	18	R	26	R	34	R	42	R	50	R	58
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
R	3	R	11	R	19	R	27	R	35	R	43	R	51	R	59
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
R	4	R	12	R	20	R	28	R	36	R	44	R	52	R	60
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
R	5	R	13	R	21	R	29	R	37	R	45	R	53	R	61
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
R	6	R	14	R	22	R	30	R	38	R	46	R	54	R	62
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
R	7	R	15	R	23	R	31	R	39	R	47	R	55	R	63
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

Figure 7-69 — The decoding order of transitional BAB (bab_size==16)

7.10.6.2.6 Decoding of exceptional BAB

The exceptional BAB coding includes two steps. First the collocated $BAB_{1/2}$ (vertically enhanced exceptional BAB with size bab_size/2 x bab_size) in the half-higher resolution layer is coded. Then the exceptional BAB in the current layer is coded based on the $BAB_{1/2}$ at half-higher resolution layer. Figure 7-70 illustrates the decoding steps for exceptional BABs.

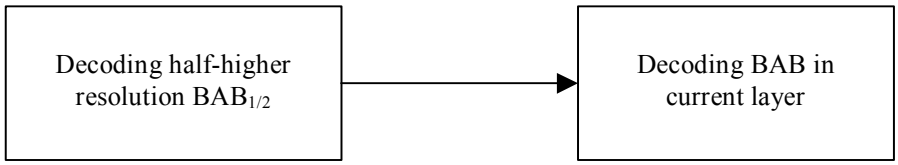


Figure 7-70 — Decoding steps for exceptional BABs

When decoding a vertically enhanced BAB, pixels from neighboring BABs in the half resolution layer can be used to make up the context. A 1 pixel wide border about the current $BAB_{1/2}$ is used as depicted in Figure 7-71.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

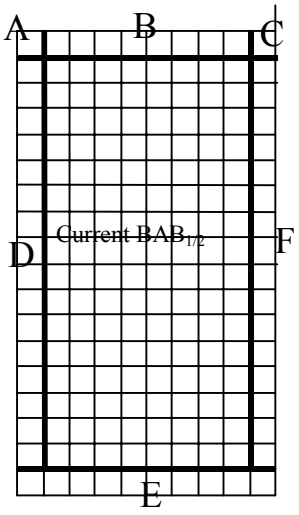


Figure 7-71 — Bordered half-higher resolution $BAB_{1/2}$.
A: TOP_LEFT_BORDER. B: TOP_BORDER. C: TOP_RIGHT_BORDER.
D: LEFT_BORDER. E: BOTTOM, F: RIGHT_BORDER

The border of the current $BAB_{1/2}$ is partitioned into 6 regions:

a) TOP_LEFT_BORDER, which contains pixels from the $BAB_{1/2}$ located to the upper-left of the current $BAB_{1/2}$ and which consists 1 pixel

b) TOP_BORDER, which contains pixels from the $BAB_{1/2}$ located above the current $BAB_{1/2}$ and which consists of 1 lines of the number of bab_size/2 pixels

c) TOP_RIGHT_BORDER, which contains pixels from the $BAB_{1/2}$ located to the upper-right of the current $BAB_{1/2}$ and which consists of 1 pixels

d) LEFT_BORDER, which contains pixels from the $BAB_{1/2}$ located to the left of the current $BAB_{1/2}$ and which consists of 1 columns of the number of bab_size pixels

e) BOTTOM_BORDER, which contains pixels from the collocated pixels in the lower layer and which consists of 1 lines in bottom of (bab_size/2+2) pixels. The pixels at (2i,j) in this border are filled by of the pixels at (i, j) in lower layer.

f) RIGHT_BORDER, which contains pixels from the collocated pixels in the lower layer and which consists of 1 column of the number of bab_size pixels. The pixel at (2i,j) in this border is filled by pixel at (i,j) from lower layer. And the pixel at (2i+1, j) in this border is filled by (i,j) from lower layer if the pixel at (i,j-1) in lower layer is not zero, otherwise it is filled by the logic OR of pixel at (i,j) and pixel at (i+1,j) in the lower layer.

By using 8-bits contexts, a pixel is arithmetic decoded as in CAE. The contexts for exceptional BAB decoding of shape object are described in Figure 7-72 (a), (b) and Figure 7-73 (a), (b). The contexts are calculated as below. And for each context, the probability table, sto_enh_odd_prob0[], sto_enh_even_prob0[], sto_enh_odd_prob1[], and sto_enh_even_prob1[] are used, respectively.

Also note that all the probability tables in the Appendix give the probability of zeros under given contexts. The probability tables are normalized to within range [0, 65536]. There are three special cases,

(1) Probability of zero under certain context is zero. This means that under that context, it is certain that the coded value and can only be one.

(2) Probability of zero under certain context is 65536. This means that under that context, it is certain that the coded value can only be zero.

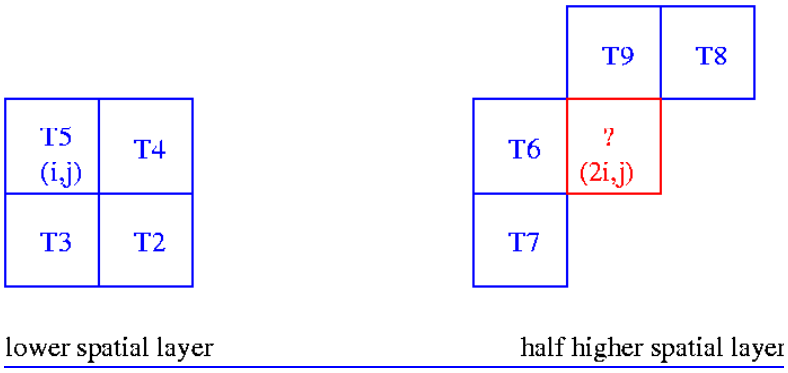
(3) Probability of zero under certain context is 65537. This means that this specific context can never occur.

$Context_0 = (T_9<<7)|\ (T_8<<6)|\ (T_7<<5)|(T_6<<4)|(T_5<<3)|(T_4<<2)|(T_3<<1)|T_2$

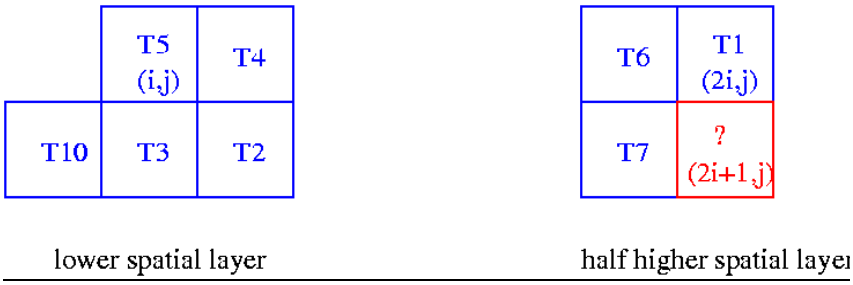
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$Context_1 = (T_1<<7) | (T_{10}<<6) | (T_7<<5) | (T_6<<4) | (T_5<<3) | (T_4<<2) | (T_3<<1) | T_2$

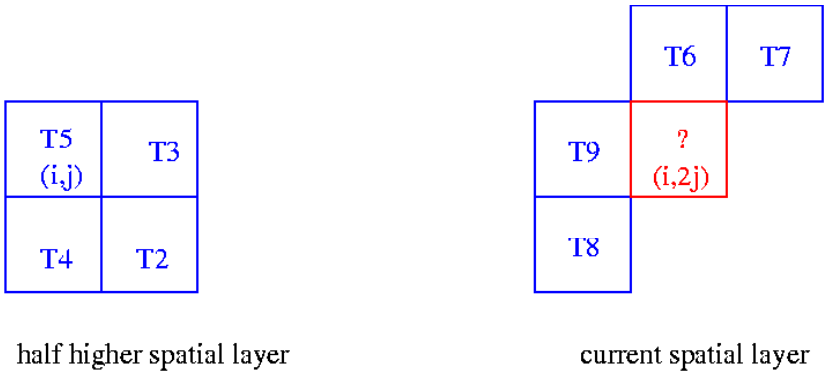


(a)

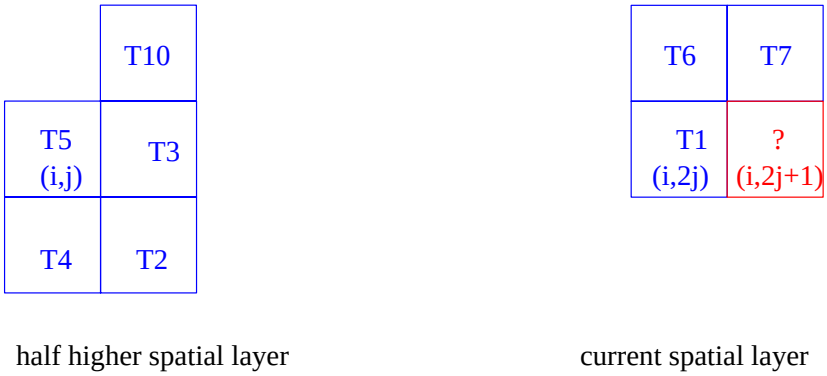


(b)

Figure 7-72 — The context used for vertically enhanced exceptional BABs



(a)



(b)

Figure 7-73 — The context used for enhanced exceptional BABs

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

7.11 Mesh object decoding

An overview of the decoding process is show in Figure 7-74.

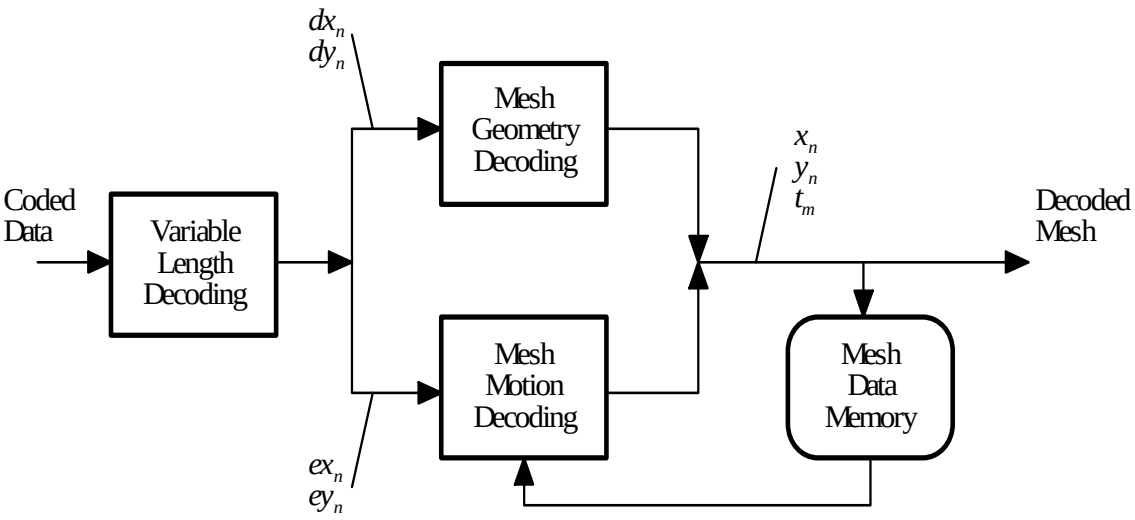


Figure 7-74 — Simplified 2D Mesh Object Decoding Process

Variable length decoding takes the coded data and decodes either node point location data or node point motion data. Node point location data is denoted by dx_n, dy_n and node point motion data is denoted by ex_n, ey_n, where n is the node point index (n = 0, ..., N-1). Next, either mesh geometry decoding or mesh motion decoding is applied. Mesh geometry decoding computes the node point locations from the location data and reconstructs a triangular mesh from the node point locations. Mesh motion decoding computes the node point motion vectors from the motion data and applies these motion vectors to the node points of the previous mesh to reconstruct the current mesh.

The reconstructed mesh is stored in the mesh data memory, so that it may be used by the motion decoding process for the next mesh. Mesh data consists of node point locations (x_n, y_n) and triangles t_m, where m is the triangle index (m = 0, ..., M-1) and each triangle t_m contains a triplet <i, j, k> which stores the indices of the node points that form the three vertices of that triangle.

A mesh object consists of a sequence of mesh object planes. The is_intra flag of the mesh object plane class determines whether the data that follows specifies the initial geometry of a new dynamic mesh, or that it specifies the motion of the previous mesh to the current mesh, in a sequence of meshes. Firstly, the decoding of mesh geometry is described; then, the decoding of mesh motion is described. In this part of ISO/IEC 14496, a pixel-based coordinate system is assumed, where the x-axis points to the right from the origin, and the y-axis points down from the origin.

7.11.1 Mesh geometry decoding

Since the initial 2D triangular mesh is either a uniform mesh or a Delaunay mesh, the mesh triangular structure (i.e. the connections between node points) is not coded explicitly. Only a few parameters are coded for the uniform mesh; only the 2D node point coordinates $\vec{p}_n = (x_n, y_n)$ are coded for the Delaunay mesh. In each case, the coded information defines the triangular structure of the mesh implicitly, such that it can be computed uniquely by the decoder. The mesh_type_code specifies whether the initial mesh is uniform or Delaunay.

7.11.1.1 Uniform mesh

A 2D uniform mesh subdivides a rectangular object plane area into a set of rectangles, where each rectangle in turn is subdivided into two triangles. Adjacent triangles share node points. The node points are spaced equidistant horizontally as well as vertically. An example of a uniform mesh is given in Figure 7-75.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Five parameters are used to specify a uniform mesh. The first two parameters, nr_mesh_nodes_hor and nr_mesh_nodes_vert, specify the number of node points of the mesh in the horizontal, resp. vertical direction. In the example of Figure 7-75, nr_mesh_nodes_hor is equal to 5 and nr_mesh_nodes_vert is equal to 4. The next two parameters, mesh_rect_size_hor and mesh_rect_size_vert, specify the horizontal, resp. vertical size of each rectangle in half pixel units. The meaning of these parameters is indicated in Figure 7-75. The last parameter, triangle_split_code, specifies how each rectangle is split to form two triangles. The four methods of splitting that are allowed are indicated in Figure 7-76. The top-left node point of a uniform mesh coincides with the origin of a local coordinate system.

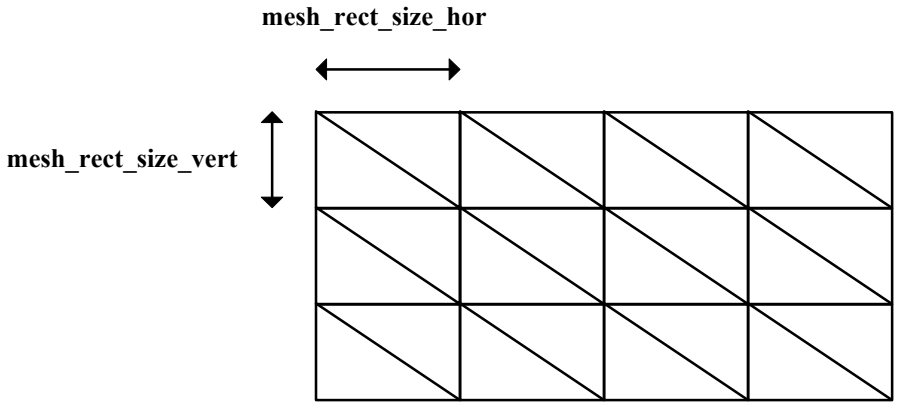


Figure 7-75 — Specification of a uniform 2D mesh

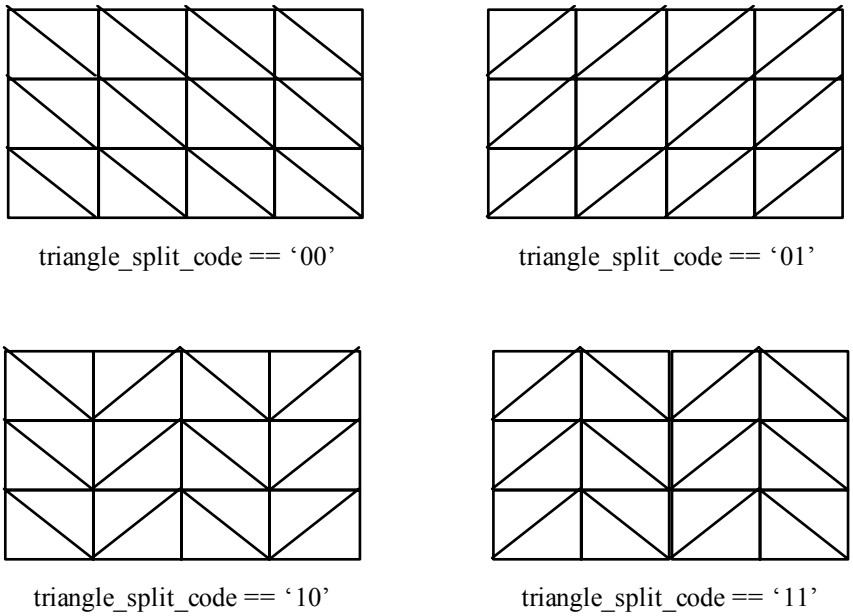


Figure 7-76 — Illustration of the types of uniform meshes defined

7.11.1.2 Delaunay mesh

First, the total number of node points in the mesh N is decoded; then, the number of node points that are on the boundary of the mesh N_b is decoded. Note that N is the sum of the number of nodes in the interior of the mesh, N_i and the number of nodes on the boundary, N_b,

$$N = N_i + N_b .$$

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Now, the locations of boundary and interior node points are decoded, where we assume the origin of the local coordinate system is at the top left of the bounding rectangle surrounding the initial mesh. The x-, resp. y-coordinate of the first node point, $\vec{p}_0 = (x_0, y_0)$, is decoded directly, where x_0 and y_0 are specified w.r.t. to the origin of the local coordinate system. All the other node point coordinates are computed by adding a dx_n, resp. dy_n value to, resp. the x- and y-coordinate of the previously decoded node point. Thus, the coordinates of the initial node point $\vec{p}_0 = (x_0, y_0)$ is decoded as is, whereas the coordinates of all other node points, $\vec{p}_n = (x_n, y_n)$, n = 1, ..., N - 1, are obtained by adding a decoded value to the previously decoded node point coordinates:

$$x_n = x_{n-1} + dx_n \quad \text{and} \quad y_n = y_{n-1} + dy_n .$$

The ordering in the sequence of decoded locations is such that the first N_b locations correspond to boundary nodes. Thus, after receiving the first N_b locations, the decoder is able to reconstruct the boundary of the mesh by connecting each pair of successive boundary nodes, as well as the first and the last, by straight-line edge segments. The next N - N_b values in the sequence of decoded locations correspond to interior node points. Thus, after receiving N nodes, the locations of both the boundary and interior nodes can be reconstructed, in addition to the polygonal shape of the boundary. This is illustrated with an example in Figure 7-77.

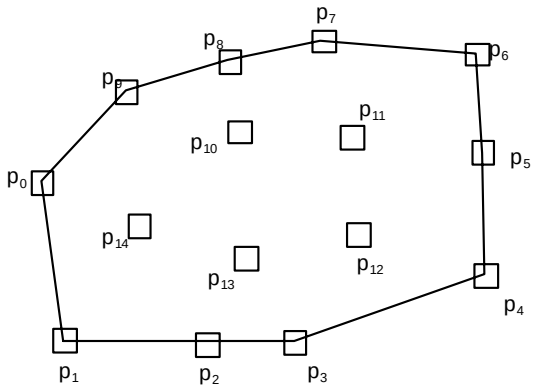


Figure 7-77 — Decoded node points and mesh boundary edge

The mesh is finally obtained by applying constrained Delaunay triangulation to the set of decoded node points, where the polygonal mesh boundary is used as a constraint. A constrained triangulation of a set of node points $\vec{p}_n$ contains the line segments between successive node points on the boundary as edges and contains triangles only in the interior of the region defined by the boundary. Each triangle $t_k = \langle \vec{p}_l, \vec{p}_m, \vec{p}_n \rangle$ of a constrained Delaunay triangulation furthermore satisfies the property that the circumcircle of t_k does not contain in its interior any node point $\vec{p}_r$ visible from all three vertices of t_k. A node point is visible from another node point if a straight line drawn between them falls entirely inside or exactly on the constraining polygonal boundary. The Delaunay triangulation process is defined as any algorithm that is equivalent to the following.

a. Determine any triangulation of the given node points such that all triangles are contained in the interior of the polygonal boundary. The triangulation shall contain 2 N_i + N_b – 2 triangles.

b. Inspect each interior edge, shared by two opposite triangles, of the triangulation and test if the edge is locally Delaunay. If there is an interior edge that is not locally Delaunay, the two opposite triangles <p_a, p_b, p_c> and <p_a, p_c, p_d> sharing this edge are replaced by triangles <p_a, p_b, p_d> and <p_b, p_c, p_d>. Continue until all interior edges of the triangulation are locally Delaunay.

An interior edge, shared by two opposite triangles <p_a, p_b, p_c> and <p_a, p_c, p_d>, is locally Delaunay if point p_d is outside the circumcircle of triangle <p_a, p_b, p_c>. If point p_d is inside the circumcircle of triangle <p_a, p_b, p_c>, then the edge is not locally Delaunay. If point p_d is exactly on the circumcircle of triangle <p_a, p_b, p_c>, then the edge between points p_a and p_c is deemed locally Delaunay only if point p_b or point p_d is the point (among these four points) with the maximum x-coordinate, or, in case there is more than one point with the same maximum x-coordinate, the point with the maximum y-coordinate among these points.An example of a mesh obtained by constrained triangulation of the node points of Figure 7-77 is shown in Figure 7-78.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

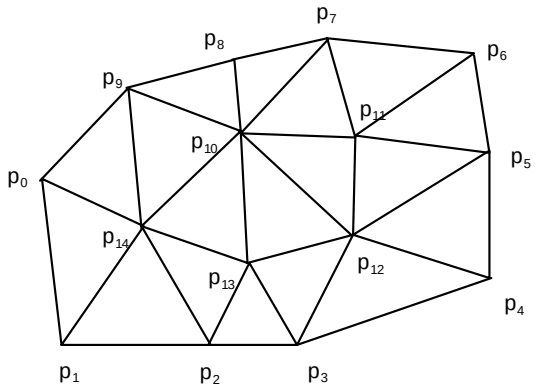


Figure 7-78 — Decoded triangular mesh obtained by constrained Delaunay triangulation

7.11.2 Decoding of mesh motion vectors

Each node point $\vec{p}_n$ of a 2D Mesh Object Plane numbered *k* in the sequence of Mesh Object Planes has a 2D motion vector $\vec{v}_n = (vx_n, vy_n)$, defined from Mesh Object Plane *k* to *k*+1. By decoding these motion vectors, one is able to reconstruct the locations of node points in Mesh Object Plane numbered *k*+1. The triangular topology of the mesh remains the same throughout the sequence. Node point motion vectors are decoded according to a predictive method, i.e., the components of each motion vector are predicted using the components of already decoded motion vectors of other node points.

7.11.2.1 Motion vector prediction

To decode the motion vector of a node point $\vec{p}_n$ that is part of a triangle $t_k = \langle \vec{p}_l, \vec{p}_m, \vec{p}_n \rangle$, where the two motion vectors vectors $\vec{v}_l$ and $\vec{v}_m$ of the nodes $\vec{p}_l$ and $\vec{p}_m$ have already been decoded, one can use the values of $\vec{v}_l$ and $\vec{v}_m$ to predict $\vec{v}_n$ and add the prediction vector to a decoded prediction error vector. Starting from an initial triangle t_k of which all three node motion vectors have been decoded, there must be at least one other, neighboring, triangle t_w that has two nodes in common with t_k. Since the motion vectors of the two nodes that t_k and t_w have in common have already been decoded, one can use these two motion vectors to predict the motion vector of the third node in t_w. The actual prediction vector $\vec{w}_n$ is computed by averaging of the two prediction motion vectors and the components of the prediction vector are rounded to half-pixel accuracy, as follows:

$$\vec{w}_n = 0.5 \bullet (\text{floor}(vx_m + vx_l + 0.5), \text{floor}(vy_m + vy_l + 0.5)),$$
$$\vec{v}_n = \vec{w}_n + \vec{e}_n$$

Here, $\vec{e}_n = (ex_n, ey_n)$ denotes the prediction error vector, the components of which are decoded from variable length codes. This procedure is repeated while traversing the triangles and nodes of the mesh, as explained below. While visiting all triangles of the mesh, the motion vector data of each node is decoded from the bitstream one by one. Note that no prediction is used to decode the first motion vector,

$$\vec{v}_{n_0} = \vec{e}_{n_0},$$

and that only the first decoded motion vector is used as a predictor to code the second motion vector,

$$\vec{v}_{n_1} = \vec{v}_{n_0} + \vec{e}_{n_1}.$$

Note further that the prediction error vector is specified only for node points with a nonzero motion vector. For all other node points, the motion vector is $\vec{v}_n = (0,0)$.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Finally, the horizontal and vertical components of mesh node motion vectors are processed to lie within a certain range, equivalent to the processing of video block motion vectors described in subclause 7.6.3.

7.11.2.2 Mesh traversal

We use a *breadth-first traversal* to order all the triangles and nodes in the mesh numbered *k*, and to decode the motion vectors defined from mesh *k* to *k*+1. The breadth-first traversal is determined uniquely by the topology and geometry of an intra-coded mesh. That is, the ordering of the triangles and nodes shall be computed on an intra-coded Mesh Object Plane and remains constant for the following predictive-coded Mesh Object Planes. The breadth-first traversal of the mesh triangles is defined as follows (see Figure 7-79 for an illustration).

First, define the *initial triangle* as follows. Define the top left mesh node as the node *n* with minimum $x_n + y_n$, assuming the origin of the local coordinate system is at the top left. If there is more than one node with the same value of $x_n + y_n$, then choose the node point among these with minimum *y*. The initial triangle is the triangle that contains the edge between the top-left node of the mesh and the next clockwise node on the boundary. Label the initial triangle with the number 0.

Next, all other triangles are iteratively labeled with numbers 1, 2, ..., *M* - 1, where *M* is the number of triangles in the mesh, as follows.

> Among all labeled triangles that have adjacent triangles which are not yet labeled, find the triangle with the lowest number label. This triangle is referred to in the following as the *current triangle*. Define the *base edge* of this triangle as the edge that connects this triangle to the already labeled neighboring triangle with the lowest number. In the case of the initial triangle, the base edge is defined as the edge between the top-left node and the next clockwise node on the boundary. Define the *right edge* of the current triangle as the next counterclockwise edge of the current triangle with respect to the base edge; and define the *left edge* as the next clockwise edge of the current triangle with respect to the base edge. That is, for a triangle $t_k = \langle \vec{p}_l, \vec{p}_m, \vec{p}_n \rangle$, where the vertices are in clockwise order, if $\langle \vec{p}_l \vec{p}_m \rangle$ is the base edge, then $\langle \vec{p}_l \vec{p}_n \rangle$ is the right edge and $\langle \vec{p}_m \vec{p}_n \rangle$ is the left edge.
>
> Now, check if there is an unlabeled triangle adjacent to the current triangle, sharing the right edge. If there is such a triangle, label it with the next available number. Then check if there is an unlabeled triangle adjacent to the current triangle, sharing the left edge. If there is such a triangle, label it with the next available number.

This process is continued iteratively until all triangles have been labeled with a unique number *m*.

The ordering of the triangles according to their assigned label numbers implicitly defines the order in which the motion vector data of each node point is decoded, as described in the following. Initially, motion vector data for the top-left node of the mesh is retrieved from the bitstream. No prediction is used for the motion vector of this node, hence this data specifies the motion vector itself. Then, motion vector data for the second node, which is the next clockwise node on the boundary w.r.t. the top-left node, is retrieved from the bitstream. This data contains the prediction error for the motion vector of this node, where the motion vector of the top-left node is used as a prediction. Mark these first two nodes (that form the base edge of the initial triangle) with the label ‘done’.

Next, process each triangle as determined by the label numbers. For each triangle, the base edge is determined as defined above. The motion vectors of the two nodes of the base edge of a triangle are used to form a prediction for the motion vector of the third node of that triangle. If that third node is not yet labeled ‘done’, motion vector data is retrieved and used as prediction error values, i.e. the decoded values are added to the prediction to obtain the actual motion vector. Then, that third node is labeled ‘done’. If the third note is already labeled ‘done’, then it is ignored and no data is retrieved. Note that due to the ordering of the triangles as defined above, the two nodes on the base edge of a triangle are guaranteed to be labeled ‘done’ when that triangle is processed, signifying that their motion vectors have already been decoded and may be used as predictors.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

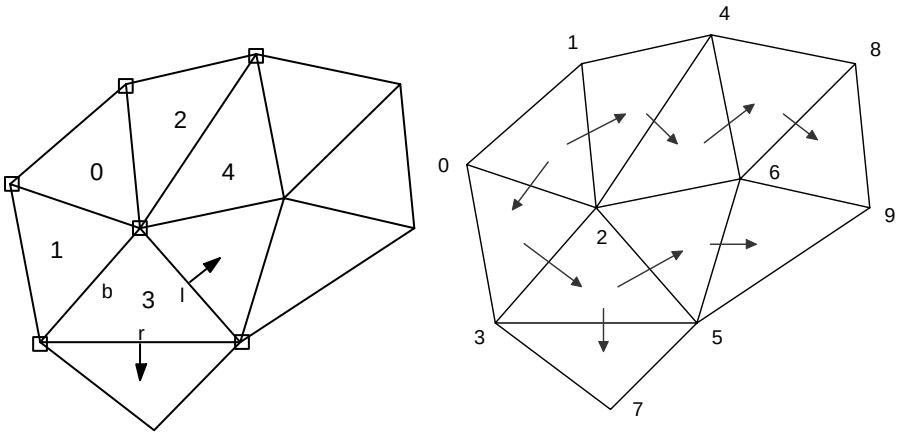


Figure 7-79 — Breadth-first traversal of a 2D triangular example mesh

In Figure 7-79 an example is shown of breadth-first traversal. On the left, the traversal is halfway through the mesh - five triangles have been labeled (with numbers) and the motion vectors of six node points have been decoded (marked with a box symbol). The triangle which has been labeled '3' is the 'current triangle'; the base edge is 'b'; the right and left edge are 'r' and 'l'. The triangles that will be labeled next are the triangles sharing the right, resp. left edge with the current triangle. After those triangles are labeled, the triangle which has been labeled '4' will be the next 'current triangle' and another motion vector will be decoded. On the right, the traversed 2D triangular mesh is shown, illustrating the transitions between triangles and final order of node points according to which respective motion vectors are decoded.

7.12 FBA object decoding

The frame rate, time code and skip frames information is independently associated with face, body or both depending on the fba_object_mask. For example, if the frame rate is set to 30Hz during a frame with fba_object_mask='01' (face) followed by a frame with frame rate set to 15Hz with fba_object_mask='10' (body) then the frame rate for the face remains set to 30Hz. If the fba_object_mask='11' the temporal information is applied to both face and body.

In frame based decoding, at the beginning of each predictive frame, the adaptive arithmetic coder shall be reset. At the end of each frame, it shall be flushed.

Each animated field has its own set of models. At each intra frame, if the stream has been declared in random access mode (see ISO/IEC 14496-1), the models shall be reset to the uniform statistics. If the stream is not in random access mode, the models shall not be reset unless the decoding structures (AnimQP) are modified.

For more details on the arithmetic decoding process, refer to ISO/IEC 14496-1, subclause 9.3.8.

7.12.1 Frame based face object decoding

This subclause specifies the additional decoding process required for face object decoding.

The coded data is decoded by an arithmetic decoding process. The arithmetic decoding process is described in detail in Annex B. Following the arithmetic decoding, the data is de-quantised by an inverse quantisation process. The FAPs are obtained by a predictive decoding scheme as shown in Figure 7-80.

The base quantisation step size QP for each FAP is listed in Table C-1. The quantiser scale fap_quant is applied uniformly to all FAPs. The magnitude of the quantisation scaling factor ranges from 1 to 8. The value of fap_quant == 0 has a special meaning, it is used to indicate lossless coding mode, so no Inverse quantisation is applied. The quantisation step size is obtained as follows:

```
if (fap_quant)
    qstep = QP * fap_quant
else
    qstep = 1
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The inverse quantised FAP'(t) is obtained from the decoded coefficient FAP''(t) as follows:

FAP'(t) = qstep * FAP''(t)

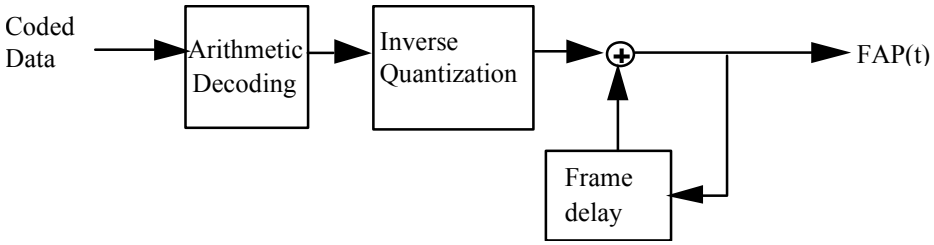


Figure 7-80 — FAP decoding

7.12.1.1 Decoding of faps

For a given frame FAPs in the decoder assume one of three of the following states:

- set by a value transmitted by the encoder
- retain a value previously sent by the encoder
- interpolated by the decoder

FAP values which have been initialized in an intra coded FAP set are assumed to retain those values if subsequently masked out unless a special mask mode is used to indicate interpolation by the decoder. FAP values which have never been initialized must be estimated by the decoder. For example, if only FAP group 2 (inner lip) is used and FAP group 8 (outer lip) is never used, the outer lip points must be estimated by the decoder. In a second example the FAP decoder is also expected to enforce symmetry when only the left or right portion of a symmetric FAP set is received (e.g. if the left eye is moved and the right eye is subject to interpolation, it is to be moved in the same way as the left eye).

7.12.2 DCT based face object decoding

The bitstream is decoded into segments of FAPs, where each segment is composed of a temporal sequence of 16 FAP object planes. The block diagram of the decoder is shown in Figure 7-81.

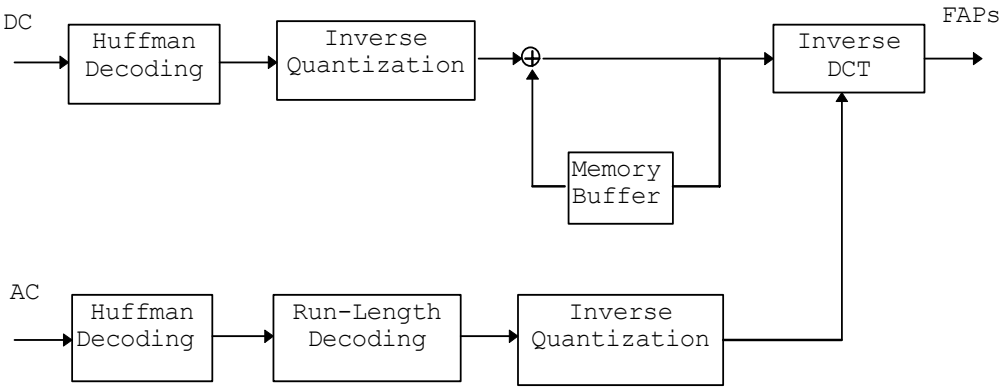


Figure 7-81 — Block diagram of the DCT-based decoding process

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The DCT-based decoding process consists of the following three basic steps:

- Differential decoding the DC coefficient of a segment.
- Decoding the AC coefficients of the segment.
- Determining the 16 FAP values of the segment using inverse discrete cosine transform (IDCT).

A uniform quantisation step size is used for all AC coefficients. The quantisation step size for AC coefficients is obtained as follows:

```
qstep[i]  = fap_scale[fap_quant_inex] * DCTQP[i]
```

where DCTQP[i] is the base quantisation step size and its value is defined in subclause 6.3.10.10. The quantisation step size of the DC coefficient is one-third of the AC coefficients. Different quantisation step sizes are used for different FAPs.

The DCT-based decoding process is applied to all FAP segments except the viseme (FAP #1) and expression (FAP #2) parameters. The latter two parameters are differential decoded without transform. The decoding of viseme and expression segments are described at the end of this subclause.

For FAP #3 to FAP #68, the DC coefficient of an intra coded segment is stored as a 16-bit signed integer if its value is within the 16-bit range. Otherwise, it is stored as a 31-bit signed integer. For an inter coded segment, the DC coefficient of the previous segment is used as a prediction of the current DC coefficient. The prediction error is decoded using a Huffman table of 512 symbols. An "ESC" symbol, if obtained, indicates that the prediction error is out of the range [-255, 255]. In this case, the next 16 bits extracted from the bitstream are represented as a signed 16-bit integer for the prediction error. If the value of the integer is equal to -256*128, it means that the value of the prediction error is over the 16-bit range. Then the following 32 bits from the bitstream are extracted as a signed 32-bit integer, in twos complement format and the most significant bit first.

The AC coefficients, for both inter and intra coded segments, are decoded using Huffman tables. The run-length code indicates the number of leading zeros before each non-zero AC coefficient. The run-length ranges from 0 to 14 and proceeds the code for the AC coefficient. The symbol 15 in the run length table indicates the end of non-zero symbols in a segment. Therefore, the Huffman table of the run-length codes contains 16 symbols. The values of non-zero AC coefficients are decoded in a way similar to the decoding of DC prediction errors but with a different Huffman table.

The bitstreams corresponding to viseme and expression segments are differential decoded without IDCT. For an intra coded segment, the quantised values of the first viseme_select1, viseme_select2, viseme_blend, expression_select1, expression_select2, expression_intensity1, and expression_intensity2 within the segment are decoded using fixed length code. These first values are used as the prediction for the second viseme_select1, viseme_select2, … etc of the segment and the prediction error are differential decoded using Huffman tables. For an inter coded segment, the last viseme_select1, for example, of the previous decoded segment is used to predict the first viseme_select1 of the current segment. The decoded values (before inverse quantisation) of differential coded viseme and expression parameter fields are obtained.

```
byviseme_segment_select1q[k] = viseme_segment_select1q[k-1] +
    viseme_segment_select1q_diff[k] - 14
viseme_segment_select2q[k]       = viseme_segment_select2q[k-1] +
    viseme_segment_select2q_diff[k] - 14
viseme_segment_blendq[k]     = viseme_segment_blendq[k-1] +
    viseme_segment_blendq_diff[k] - 63
expression_segment_select1q[k]   = expression_segment_select1q[k-1] +
    expression_segment_select1q_diff[k] - 6
expression_segment_select2q[k]   = expression_segment_select2q[k-1] +
    expression_segment_select2q_diff[k] - 6
expression_segment_intensity1q[k] = expression_segment_intensity1q[k-1] +
    expression_segment_intensity1q_diff[k] - 63
expression_segment_intensity2q[k] = expression_segment_intensity2q[k-1] +
    expression_segment_intensity2q_diff[k] - 63
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.12.3 Decoding of the viseme parameter fap 1

Fourteen visemes have been defined for selection by the Viseme Parameter FAP 1, the definition is given in Annex C. The viseme parameter allows two visemes from a standard set to be blended together. The viseme parameter is composed of a set of values as follows.

Table 7-21 — Viseme parameter range

viseme () {	Range
viseme_select1	0-14
viseme_select2	0-14
viseme_blend	0-63
viseme_def	0-1
}	

Viseme_blend is quantised (step size = 1) and defines the blending of viseme1 and viseme2 in the decoder by the following symbolic expression where viseme1 and 2 are graphical interpretations of the given visemes as suggested in the non-normative annex.

final viseme = (viseme 1) * (viseme_blend / 63) + (viseme 2) * (1 - viseme_blend / 63)

The viseme can only have impact on FAPs that are currently allowed to be interpolated.

If the viseme_def bit is set, the current mouth FAPs can be used by the decoder to define the selected viseme in terms of a table of FAPs. This FAP table can be used when the same viseme is invoked again later for FAPs which must be interpolated.

7.12.4 Decoding of the viseme parameter fap 2

The expression parameter allows two expressions from a standard set to be blended together.The expression parameter is composed of a set of values as follows.

Table 7-22 — Expression parameter range

expression () {	Range
expression_select1	0-6
expression_intensity1	0-63
expression_select2	0-6
expression_intensity2	0-63
init_face	0-1
expression_def	0-1
}	

Expression_intensity1 and expression_intensity2 are quantised (step size = 1) and define excitation of expressions 1 and 2 in the decoder by the following equations where expressions 1 and 2 are graphical interpretations of the given expression as suggested by the non-normative reference:

final expression = expression1 * (expression_intensity1 / 63)+ expression2 * (expression_intensity2 / 63)

The decoder displays the expressions according to the above fomula as a superposition of the 2 expressions.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The expression can only have impact on FAPs that are currently allowed to be interpolated. If the init_face bit is set, the neutral face may be modified within the neutral face constraints of mouth closure, eye opening, gaze direction, and head orientation before FAPs 3-68 are applied. If the expression_def bit is set, the current FAPs can be used to define the selected expression in terms of a table of FAPs. This FAP table can then be used when the same expression is invoked again later.

7.12.5 Fap masking

The face is animated by sending a stream of facial animation parameters. FAP masking, as indicated in the bitstream, is used to select FAPs. FAPs are selected by using a two level mask hierarchy. The first level contains two bit code for each group indicating the following options:

- no FAPs are sent in the group
- a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain their previous value if any previously set value (not interpolated by decoder if previously set)
- a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain must be interpolated by the decoder
- all FAPs in the group are sent.

7.12.6 Frame Based Body Decoding

This clause specifies the additional decoding process required for body model decoding.

The BAPs are quantised and coded by a predictive coding scheme, similar to the FAPs. For each parameter to be coded in the current frame, the decoded value of this parameter in the previous frame is used as the prediction. Then the prediction error, i.e., the difference between the current parameter and its prediction, is computed and coded by arithmetic coding. This predictive coding scheme prevents the coding error from accumulating. The arithmetic decoding process is described in detail in subclause B.2.3.

Similar to FAPs, each BAP has a different precision requirement. Therefore different quantisation step sizes are applied to the BAPs. The base quantisation step size for each BAP is defined in the tables below. The bit rate is controlled by adjusting the quantisation step via the use of a quantisation index parameter called **bap_pred_quant_index** is applied uniformly to all BAPs. The magnitude of the quantisation index parameter ranges from 0 to 31 and is an index to a bap_pred_scale_table as described in subclause 6.3.10.1 to obtain the BAP_QUANT parameter.

The base quantisation step size BQP for each BAP is listed in Annex C. The quantiser scale BAP_QUANT is applied uniformly to all BAPs. The magnitude of the quantisation scaling factor ranges from 1 to 31. The value of BAP_QUANT == 0 has a special meaning, it is used to indicate lossless coding mode, so no inverse quantisation is applied. The quantisation step size is obtained as follows:

```
BAP_QUANT = bap_pred_scale[bap_pred_quant_index]
If (BAP_QUANT)
    qstep = BQP * BAP_QUANT
Else
    qstep = 1
```

The inverse quantised BAP'(t) is obtained from the decoded coefficient BAP''(t) as follows:

BAP'(t) = qstep * BAP''(t)

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

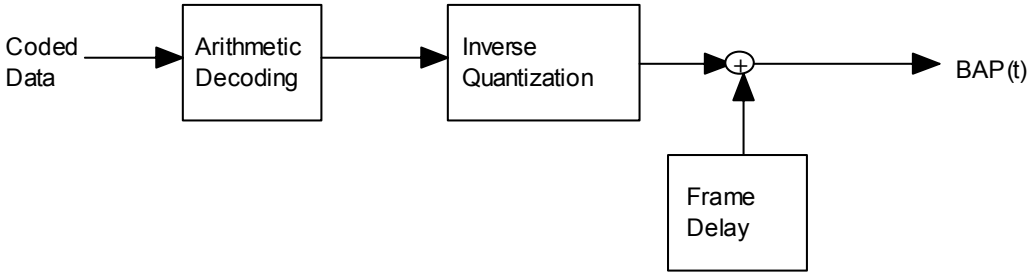


Figure 7-82 — BAP predictive coding

7.12.6.1 Decoding of BAPs

For a given frame, BAPs in the decoder assume one of the three following states:

1. set by a value transmitted by the encoder

2. retain a value previously sent by the encoder

BAP values which have been initialized in an intra coded BAP set are assumed to retain those values if subsequently masked out.

7.12.7 DCT based body object decoding

The bitstream is decoded into segments of BAPs, where each segment is composed of a temporal sequence of 16 BAP object planes. The block diagram of the decoder is shown in Figure 7-83.

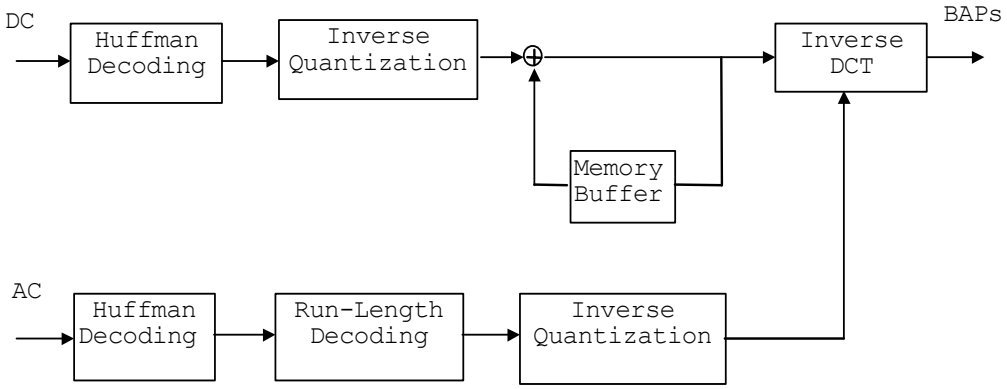


Figure 7-83 — Block diagram of the DCT-based BAPs decoding process

The DCT-based decoding process consists of the following three basic steps:

1. Differential decoding the DC coefficient of a segment.

2. Decoding the AC coefficients of the segment

3. Determining the 16 BAP values of the segment using inverse discrete cosine transform (IDCT).

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A uniform quantisation step size is used for all AC coefficients. The quantisation step size for AC coefficients is obtained as follows:

qstep[i] = bap_scale[bap_quant_inex] * BQP[i]

where BQP[i] is the base quantisation step size and its value is defined in Subclause 6.3.10.8. The quantisation step size of the DC coefficient is one-third of the AC coefficients.

The DCT-based decoding process is applied to all BAPs. The DC coefficient of an intra coded segment is stored as a 16-bit signed integer if its value is within the 16-bit range. Otherwise, it is stored as a 31-bit signed integer. For an inter-coded segment, the DC coefficient of the previous segment is used as a prediction of the current DC coefficient. The prediction error is decoded using a Huffman table of 512 symbols. An "ESC" symbol, if obtained, indicates that the prediction error is out of the range [-255, 255]. In this case, the next 16 bits extracted from the bitstream are represented as a signed 16-bit integer for the prediction error. If the value of the integer is equal to -256*128, it means that the value of the prediction error is over the 16-bit range. Then the following 32 bits from the bitstream are extracted as a signed 32-bit integer, in twos complement format and the most significant bit first.

The AC coefficients, for both inter and intra coded segments, are decoded using Huffman tables. The run-length code indicates the number of leading zeros before each non-zero AC coefficient. The run-length ranges from 0 to 14 and proceeds the code for the AC coefficient. The symbol 15 in the run length table indicates the end of non-zero symbols in a segment. Therefore, the Huffman table of the run-length codes contains 16 symbols. The values of non-zero AC coefficients are decoded in a way similar to the decoding of DC prediction errors but with a different Huffman table.

7.13 3D Mesh Object Decoding

The Topological Surgery decoder is composed of four main modules, as shown in Figure 7-84, namely:

An arithmetic decoder, which reads a section of the input stream and outputs a bit stream.

A vertex graph decoder, which reads a section of the bit stream and outputs a bounding loop look-up table. However, when has_stitches flag is on, stitches are decoded and a vertex clustering array is built.

A triangle tree decoder, which reads a section of the bit stream and outputs the length of each run and the size of each sub-tree of the triangle tree.

A triangle data decoder, which reads a section of the input bit and outputs a stream of triangle data. This stream of triangle data contains the geometry and the properties associated with each triangle.

When a 3D mesh bitstream is received, the control information including stitches and partition types, as well as the header information including properties are obtained through the demultiplexer. The remaining bitstream is fed into the arithmetic decoder.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

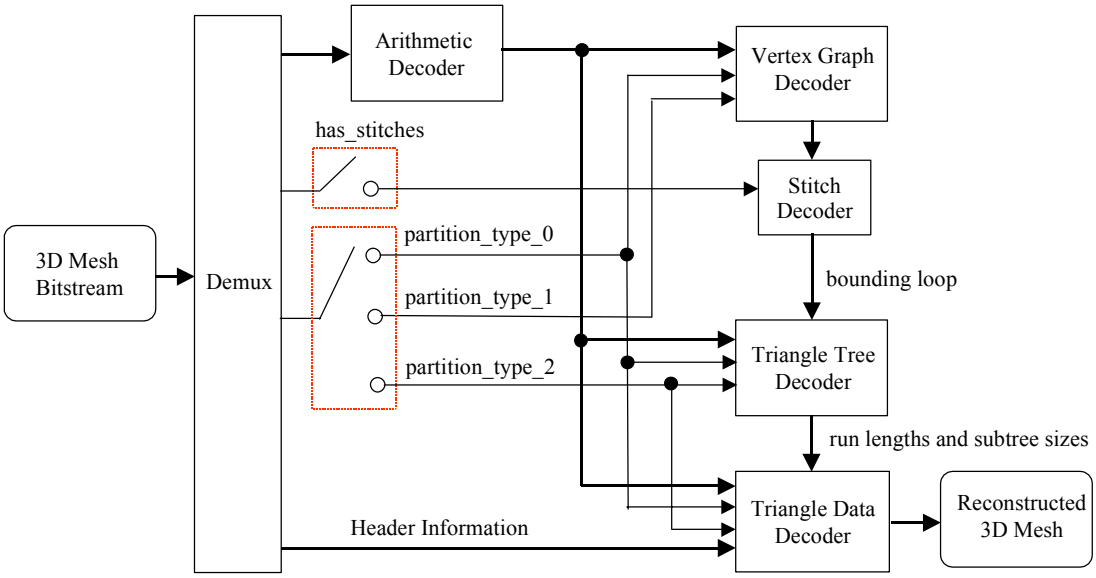


Figure 7-84 — Block diagram of the Topological Surgery decoder

In the hierarchical mode, the Topological Surgery decoder is used to decode the first (coarsest) level of detail mesh (base mesh), and the Forest Split decoder is used to recursively refine each level of detail to produce a new, higher resolution, level of detail. The Forest Split decoder is composed of four blocks, as shown in Figure 7-85, namely:

An arithmetic decoder, which reads a section of the input stream and outputs a bit stream.

A forest decoder, which reads a section of the bit stream and determines which edges of the current level mesh are part of the forest of edges.

A simple polygon decoder, which reads a section of the bit stream and outputs a sequence of simple polygons, including not only connectivity information, but also vertex coordinates, and property values.

A forest split operator, which takes the output data produced by the forest decoder and the simple polygon decoder, and applies the corresponding forest split operation, producing a new level of detail mesh.

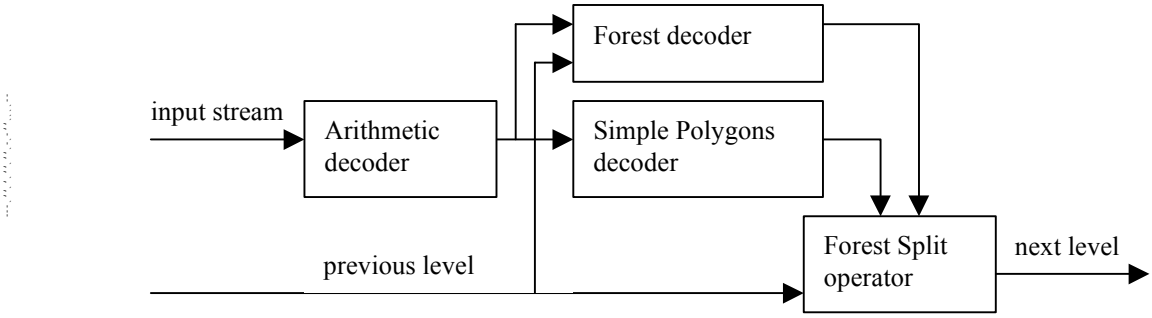


Figure 7-85 — Block diagram of the Forest Split decoder

7.13.1 Start codes and bit stuffing

A start code is a two-byte code of the form '0000 0000 00xx xxxx'. Several such codes are inserted into the bitstream for synchronisation purposes. To prevent any wrongful synchronisation, such codes shall not be emulated by the data. This is guaranteed by the insertion of a stuffing bit '1' after each byte-aligned sequence of eight 0's. The decoder shall skip these stuffing bits when parsing the bit stream. Note that the arithmetic coder is designed such as to never generate a synchronisation code. Therefore the bit skipping rule does not need be applied to the portions of the bit stream that contain arithmetic coded data.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.13.2 The Topological Surgery decoding process.

The connectivity of a 3D mesh is represented as a Simple Polygon (triangulated with a single boundary loop) with zero or more pairs of boundary vertices identified, and with zero or more internal edges labeled as polygon edges. When the pairs of corresponding boundary edges are identified, the edges of the resulting reconstructed mesh corresponding to boundary edges of the Simple Polygon form the Vertex Graph. The correspondence among pairs of Simple Polygon boundary edges is recovered by the decoding process from the structure of Vertex Graph, producing the Vertex Loop look-up table. The Vertex Graph is represented as (i) a rooted spanning tree, (ii) the Vertex Tree, and (iii) zero or more jump edges. The Simple Polygon is represented as (i) a rooted spanning tree, (ii) the Triangle Tree, which defines an order of traversal of the triangles, (iii) a sequence of marching patterns, and (iv) a sequence of polygon_edges. The marching patterns are used to reconstruct the triangles by marching on the left or on the right along the polygon bounding loop, starting from an initial edge called the root edge. The polygon edges are used to join or not to join triangles sharing a marching edge to form the faces of the mesh. As the triangles of the simple polygon are visited in the order of traversal of the Triangle Tree, simple polygon boundary edges are put in correspondence using the information contained in the Vertex Loop look-up table, and so, reconstructing the original connectivity.

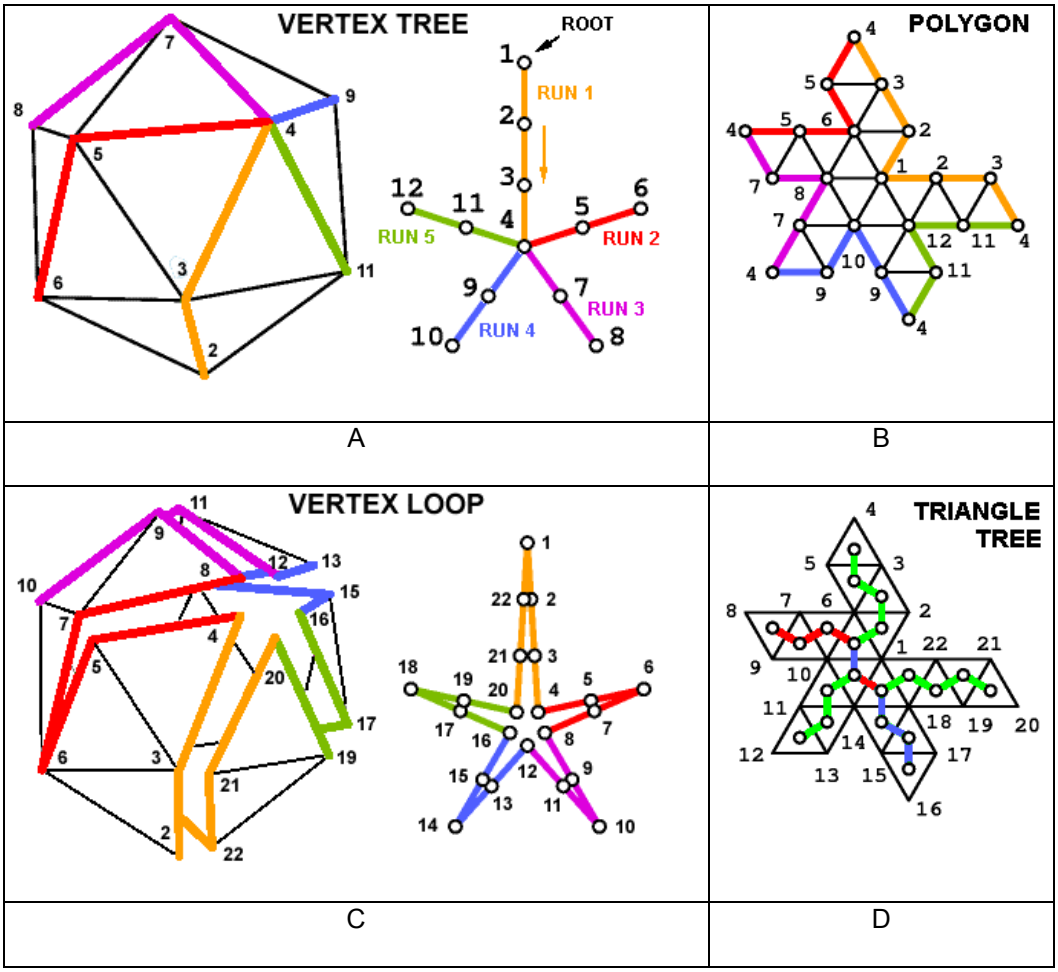


Figure 7-86 — Topological Surgery representation of a simple mesh.
A : for a simple mesh, the vertex graph is a tree.
B : when a simple mesh is cut through the vertex tree, the result is a simple polygon.
C : each edge of the vertex tree corresponds to exactly two boundary edges of the simple polygon.
D : the triangle tree spanning the dual graph of the simple polygon. In the general case, the face forest is first constructed spanning the dual graph of the 3D mesh. The vertex graph is composed of the remaining edges of the 3D mesh. If the 3D mesh has polygonal faces, they are subsequently triangulated by inserting virtual marching edges, converting the face forest into a triangle forest with one triangle tree per connected component.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

The geometry and property data are quantised and predicted as a function of ancestors in the order of traversal of the triangles. The corresponding prediction errors are transmitted in the same order of traversal, interlaced with the corresponding marching and polygon_edges, so that, as soon as all these triangle data are received, the decoder can output the corresponding triangle.

A non-manifold 3D mesh is represented as a manifold 3D mesh and a Vertex Clustering array. The Vertex Clustering array is represented in the bitstream as a sequence of stitching commands (or Stitches) with one command per vertex. The decoding process decodes the Stitches after the vertex graph and constructs the Vertex Clustering array using the Stitches. Figure 7-87 shows a 3D mesh with a singular vertex and a singular edge, with the polygons incident to the singular vertex highlighted.

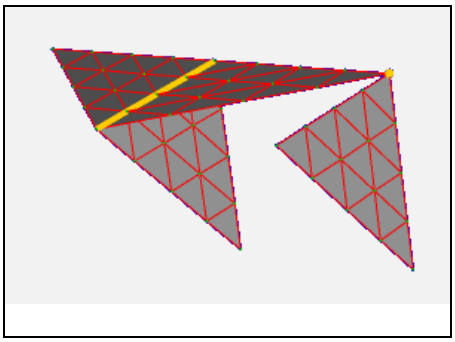


Figure 7-87 — 3D meshes with singular vertices and singular edges (each one with three incident faces) and singular edges (yellow)

At a high level, the Topological Surgery decoder performs the following steps:

```
Decode header information
for each partition {
 if (partition_type is 0) {
    for each connected component {
       decode vertex graph and construct bounding loop table
       if (has_stitches)
          decode stitches and build vertex clustering array
       decode triangle tree and construct table of triangle tree run lengths
       for each triangle in simple polygon {
          decode marching field and polygon field, and reconstruct connectivity
of triangle
          decode and reconstruct vertex coordinates and properties
       }
    }
 }
 else if (partition_type is 1) {
    for each vertex graph {
       decode vertex graph and construct bounding loop table
       if (has_stitches)
          decode stitches and build vertex clustering array
    }
 }
 else if (partition_type is 2) {
    decode partition header information
    decode triangle tree and construct table of triangle tree run lengths
    for each triangle in simple polygon {
       decode marching field, orientation field, and polygon field, and
reconstruct connectivity of triangle
       decode and reconstruct vertex coordinates and properties
    }
 }
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.13.3 The Forest Split decoding process

The hierarchical mode representation is based on the Progressive Forest split (PFS) scheme. In this scheme a 3D mesh is represented as a low resolution base mesh followed by a sequence of forest split operations. A forest split operation is represented by a forest in the graph of vertices and edges of a mesh, a sequence of simple polygons, and a sequence of vertex coordinates and property updates. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with one of the simple polygons, and finally, displacing the new vertices and properties. An optional pre-smoothing step can be used as a global predictor for the vertex coordinates, after the connectivity and properties of the new faces bound per face and per corner are updated. Also, and optional post-smoothing step can be used to reduce the quantisation artifacts after all the updates have been applied.

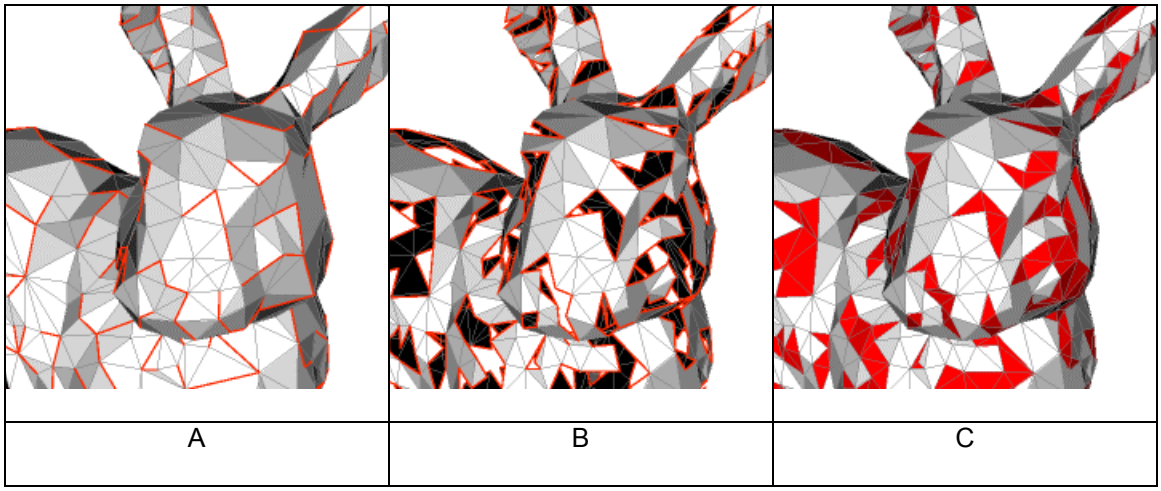


Figure 7-88 — The Forest Split operation. A 3D mesh with a forest of edges marked (A). The tree loops resulting from cutting the 3D mesh through the forest edges (B). Each tree loop is filled by a simple polygon composed of polygonal faces (C).

At a high level, the Forest Split decoder performs the following steps:

Decode pre and post-smoothing parameters.

Decode Forest.

For each tree in the forest {

Cut mesh through the tree edges, and enumerate mesh components.

Decode Triangle Tree.

Decode per triangle connectivity data and property updates

for properties bound per face and per corner.

Update connectivity by triangulating the tree loop

according to the decoded simple polygon.

}

Optionally, apply pre-smoothing step to vertex coordinates as a global predictor.

For each tree in the forest {

Decode property updates for all the tree loop vertices, faces, and corners

}

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Optionally, decode vertex coordinates, and property updates for all the vertices, faces, and corners not incident to any triangle loop.

Optionally, apply post-smoothing step to vertex coordinates.

7.13.4 Header decoder

The header information is divided into two parts. The first part contains information about the high level shading properties of the model, and the second defines the properties that are bound to the mesh.

7.13.4.1 High level shading properties

These are the **ccw**, **solid**, **convex**, **creaseAngle** fields defined in an IndexedFaceSet node. The crease angle is quantised to a 6-bit value and is reconstructed as 2*pi***creaseAngle**/63.

7.13.4.2 Property bindings and quantiser scales

There are four kinds of properties: vertex coordinates (coord), normals (normal), colors (color) and texture coordinates (texCoord). Properties can be bound to the mesh in four different ways: no binding, per vertex, per face and per corner. Not all combinations of bindings are valid. Table 7-23 lists the valid combinations. The binding of each property is obtained from coord_binding, normal_binding, color_binding and texCoord_binding, respectively.

Table 7-23 — List of valid combinations of properties and bindings

	no binding	**per vertex**	**per face**	**per corner**
coord	forbidden	valid	forbidden	forbidden
normal	valid	valid	valid	valid
color	valid	valid	valid	valid
texCoord	valid	valid	forbidden	valid

For each property for which there is a binding, i.e. the binding field does not contain the no binding value, the following information is further decoded: a bounding box, a quantisation step, a prediction mode, a list of coefficients used for linear prediction. The decoding process for the vertex coordinates is further described below. The same decoding process applies to the other properties.

The presence of a bounding box is given by the field **coord_bbox**. The bounding box is a cube represented by the position of its lower left corner and the length of its side. For geometry these parameters are given by **coord_xmin**, **coord_ymin**, **coord_zmin** and **coord_size**. If no bounding box is coded, as default bounding box is assumed. The default bounding box has its lower left corner at the origin, and a unit size.

The next field **coord_quant** indicates the number of bits to which each coordinate is quantised to. The **coord_pred_type** field indicates the prediction mode. It should always be equal to ‘10’.

The **coord_nlambda** field indicates the number of coefficients used for linear prediction. It can take one of three values, namely ’01’, ‘10’, and ‘11’. **coord_nlambda**-1 number of values are then read from the **coord_lambda** field. The last coord_lambda value is computed so that the sum of all the coord_lmabda value is equal to one. This field indicates the weight given to an ancestor for prediction. The floating point value of a coefficient is given by the decoded signed integer divided by 2 to the power **coord_quant**. The last coefficient is never transmitted and is defined to be 1 minus the sum of all other coefficients.

The are some restrictions for the normal property: the **normal_bbox** field is always false, and the **normal_quant** value must always an odd number.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.13.5 partition type

There are three partition types to convey vertext graph(vg), triangle tree(tt), and triangle data(td), The partition type specifies admissible combinations of these three pieces of data in the bitstream. See Figure 7-89.

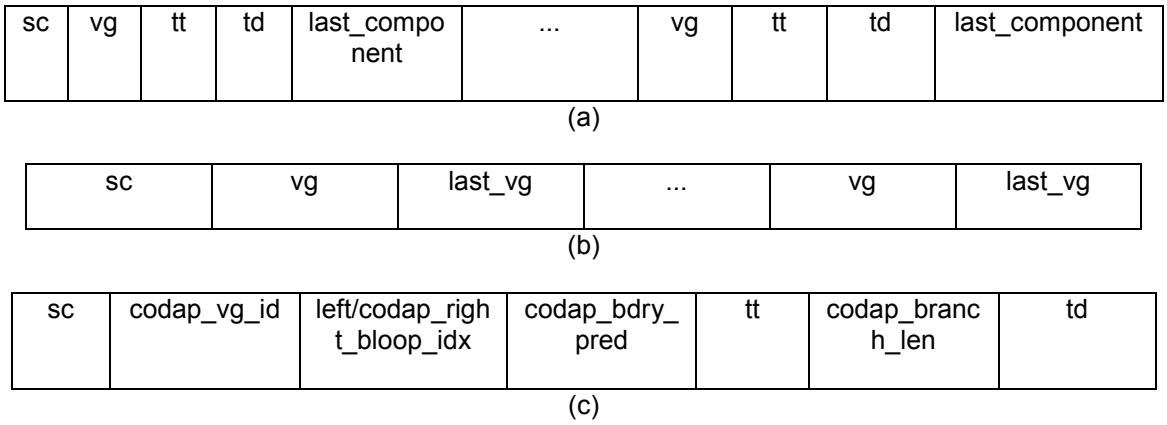


Figure 7-89 — Types of data partition : (a) one or more group of vg, tt and td included in a partition. (b) one or more vgs included in a partition. (c) the partition has one pair of tt and td.

7.13.5.1 connected component partition (partition_type_0)

This partition indicates the bitstream with one or more sequence of connected component consisting of vertex graph, triangle tree, and triangle data. The end of the connected component sequence is determined by the **last_component** field. The following partition may be either type 0 or 1.

7.13.5.2 vertex graph partition (partition_type_1)

This partition indicates the bitstream with one or more sequence of vertex graph. The end of the vertex graph sequence is determined by the **last_vg** field. The partition following the vertex graph partition shall be tt/td pair partition.

7.13.5.3 tt/td pair partition (partition_type_2)

This partition indicates the bitstream with one pair of triangle tree and triangle data. The tt/td pair partition is characterized by the vertex graph id, visiting indices, and boundary prediction mode. These variables are given fields **codap_vg_id, left/codap_right_bloop_idx**, and **codap_bdry_pred. codap_vg_id** field indicates a vertex graph corresponding to the tt/td pair. It is used to get the bounding loop information from the vertex graph. **codap_left_bloop_idx** and **codap_right_bloop_idx** fields indicate the left and right starting points of the bounding loop. They are also used to decide if a vertex in the partition is already visited in the previous partition. **codap_bdry_pred** field indicates if the vertices on the boundary of the partition should be decoded when the vertex is shared with previous partitions. If the partition ends at a branch, **codap_branch_len** field is added to the bitstream. Any type of partition may follow the tt/td pair partition.

7.13.5.3.1 Restricted boundary prediction mode (codap_bdry_pred=0)

The restricted boundary prediction mode does not duplicate vertices between partitions. Since the vertices predicted in the previous partitions may not be available, prediction is done only with the available vertices which are predicted in the current partition. If the previous partitions are lost, the triangles at the boundaries may not be reconstructed due to the fact that the vertices from the previous partitions are not available. However, the vertices predicted in the current partition may be reconstructed and decoding can continue.

The following process is applied for the prediction with a subset of all ancestors. Let the current method for prediction with 3 ancestors (*a,b,c*) be *d'* = f(*a,b,c*). Then the equation is *d* = *d'* + *e* and the value *e* is encoded. However, when the ancestors are visited in the previous partitions, the prediction method is as follows.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

if (all ancestors (*a*, *b*, and *c*) are not available) then $d' = 0$

else if (only one ancestor, say *t*, is available) then $d' = t$

else if (two ancestors, say t_1 and t_2, are available) then

if (both ancestors are edge distance 1 from current vertex) then $d' = (t_1 + t_2)/2$

else if (t_1 is edge distance 1 from current vertex) then $d' = t_1$

else $d' = t_2$

else $d' = f(a,b,c)$.

7.13.5.3.2 Extended boundary prediction mode (codap_bdry_pred=1)

When this mode is selected, the decoder assumes that every vertex is not visited in the previous partition. Thus, after the three vertices of the root triangle are predicted, the vertices in the rest of the triangles in the partition will have all three ancestors for prediction.

7.13.6 Vertex Graph Decoder

The vertex graph decoder reconstructs a sequence of vertex runs. It then constructs the Vertex Loop look-up table.

For meshes that have simple topology and that do not have a boundary, the vertex graph does not contain any loops and is hence a tree. The field **vg_simple,** if set, indicates that the graph contains no loop. Some fields below are skipped when **vg_simple** is set.

Each vertex run is characterized by several variables, namely length, last, forward, loop and loop index. These variables are given by the fields **vg_run_length**, **vg_last**, **vg_forward_run**, **vg_loop** and **loop_index**. **vg_forward_run**, **vg_loop** and **loop_index** contain information relative to loops. When **vg_simple** is true, these three fields are not coded.

The process to reconstruct the vertex loop uses an auxiliary loop queue variable, initialy empty. When **vg_loop** is set the current run is put into the loop queue. When **vg_forward_run** is not set a run is pulled out from the queue. **loop_index** determines the position of the pulled out run in the queue. **vg_forward_run** is not coded when the loop queue is empty. **loop_index** is coded by its unary representation. When the index is equal to the number of elements in the queue minus 1, the trailing bit of its unary representation is not coded. Therefore when there is a single element in the queue, **loop_index** is not coded.

The end of the graph is determined by the **vg_last** and **vg_leaf** fields. An auxiliary depth variable is used for this purpose. It is decremented each time **vg_last** is set and incremented each time **vg_leaf** is not set. The depth is initialised to zero at the beginning of the graph. The end of the graph is reached when the depth variable becomes negative.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

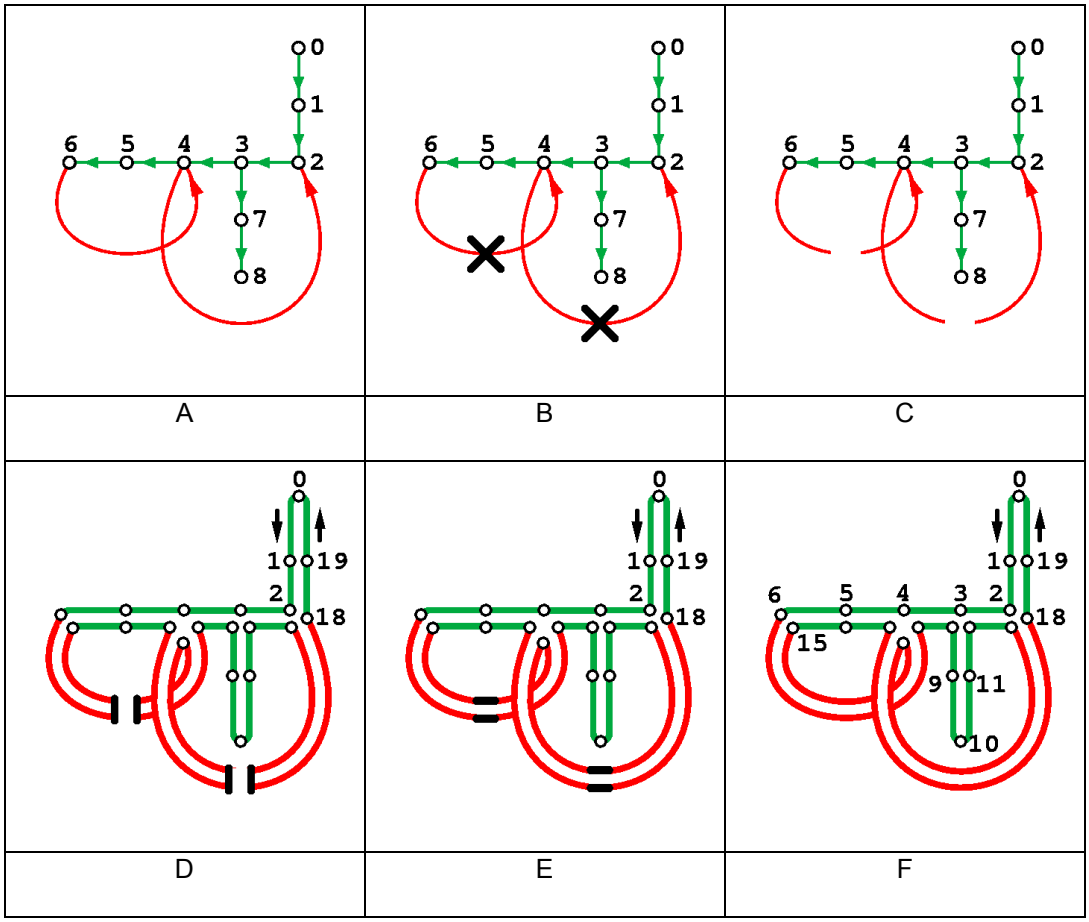


Figure 7-90 — Steps to build the bounding loop from the vertex graph.
A : vertex graph decomposed into vertex tree (green) and jump edges (red).
B : extended vertex tree is created by cutting jump edges in half.
C : the extended vertex tree has two leaves for each jump edge.
D : build the extended vertex tree loop.
E : connect the start and end of each jump edge.
F : resulting boundary loop.

Vertex numbers are assigned according to a depth-first traversal of the graph. The bounding loop look-up table is constructed by going around the graph and for each traversed vertex recording its number.

7.13.6.1 Stitches Decoder

The Stitches decoder reconstructs the Vertex Clustering that should be applied to the vertices in the Vertex Graph to reconstruct the 3D mesh. The following pseudo-code summarizes the operation of the Stitches decoder: if the boolean **has_stitches** flag in the current component is true, then for each vertex of the Vertex Graph in decoder order, the Stitches decoder decodes a stitching command from the bitstream. If the boolean value **stitching_command** is true, then the boolean value **pop_or_get** is decoded; if the boolean value **pop_or_get** is false, an unsigned integer is decoded, and associated to the current vertex i as an anchor (to stitch to). The current vertex i is then pushed to the back of the anchor_stack.

if **pop_or_get** is true, then the boolean value **pop** is decoded, followed with the unsigned integer value **stack_index**.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

```
decode(anchor_stack) {
 if (has_stitches == true)
  for (i = nV0; i < nV1; i++) {
                // nV0 is the first vertex of the current component,
                // and nV1 –1 is the last vertex
   decode stitch_cmd;
    if (stitch_cmd) {
     decode stitch_pop_or_get;
     if (stitch_pop_or_get) {
       decode stitch_pop;
       decode stitch_stitch_stack_index;
       retrieve stitching_anchor from anchor_stack;
       if (stitch_pop) {
        remove stitching_anchor from anchor_stack;
       } // end if
       decode stitch_incr_length;
       if (stitch_incr_length != 0) {
        decode stitch_incr_length_sign;
       } // end if
       decode stitch_push;
       if (stitch_push)
        push i to the back of anchor_stack
       retrieve stitch_length at anchor;
       total_length = stitch_length + stitch_incr_length;
       if (total_length > 0)
        decode stitch_reverse;
       stitch i to anchor for length of total_length
       and in a reverse fashion if (stitch_reverse);
     } // end if (stitch_pop_or_get)
     decode stitch_length;
     push i to the back of anchor_stack;
     save stitch_length at anchor i;
   } // end if (stitch_cmd)
 } // end for
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Using **stack_index**, a stitching_anchor is retrieved from the anchor_stack. This is the anchor that the current vertex i will be stitched to. If the **pop** boolean variable is true, then the stitching_anchor is removed from the anchor_stack. Then, an integer **incremental_length** is decoded as an unsigned integer. If it is different from zero, its sign (boolean **incremental_length_sign**) is decoded, and is used to update the sign of **incremental_length**. A **push_bit** boolean value is decoded. If **push_bit** is true, the current vertex i is pushed to the back of the anchor_stack. An integer **stitch_length** associated with the stitching_anchor is retrieved. A total_length is computed by adding **stitch_length** and **incremental_length**; if total_length is greater than zero, then a **reverse_bit** boolean value is decoded. Then the Vertex Clustering array is updated by stitching the current vertex i to the stitching anchor with a length equal to total_length and potentially using a reverse stitch. The decoder uses a v_father array to perform this operation. The v_father array provides for each vertex in the Vertex Graph the index of its father (predecessor) in the directed Vertex Graph without jumps (potential the vertex itself it it is a root). To stitch the current vertex i to the stitching anchor with a length equal to total_length, starting from both i and the anchor at the same time, we walk along the Vertex Graph as indicated in the v_father entries for both i and the anchor by looking up the v_father entries total_length+1 times, and for each corresponding entries ((i, anchor), (v_father[i],v_father[anchor]), (v_father[v_father[i]], v_father[v_father[anchor]])) record in the Vertex Clustering array that the entry with the largest order of traversal should be the same as the entry with the lowest order of traversal. For decoding the stitches, For instance if (j > k), then Vertex Clustering[j] = Vertex Clustering[k]. The Vertex Clustering array defines a graph that is a forest. Each time an entry in the Vertex Clustering array is changed, we perform path compression on the forest by updating the Vertex Clustering array such that each element refers directly to the root of the tree of the forest it belongs to. The following figure shows both a (forward) stitch and reverse stitch.

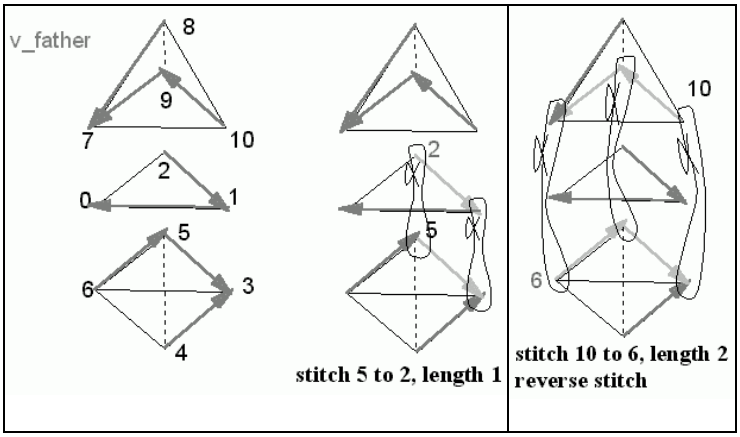


Figure 7-91 — Examples of a (forward) stitch and a reverse stitch

If the stitch is a reverse stitch, then we first follow the v_father entries starting from the anchor for a length equal to total_length (from vertex 6 to vertex 3 in the figure above), recording the intermediate vertices in a temporary array. We then follow the v_father entries starting from the vertex i and for each corresponding entry stored in the temporary array (from the last entry to the first entry), we update the Vertex Clustering array as explained above.

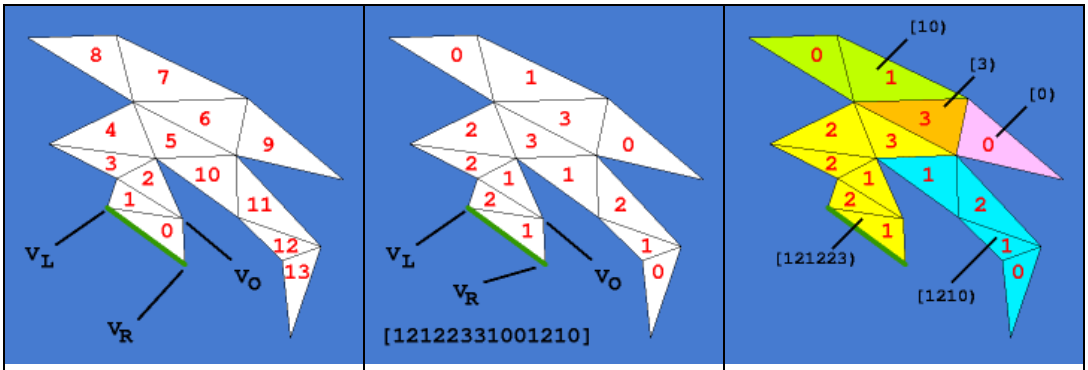
The Vertex Clustering array is used to indicate when a coord or property should be decoded or was already decoded when decoding triangle data as explained in subclause 7.13.8.

7.13.7 Triangle Tree Decoder

The triangle tree decoder reconstructs a sequence of triangle runs. For each run it generates a run length and the size of its sub-tree.

The triangle tree is a sequence of triangle runs. For each run a length **tt_run_length** and a boolean flag **tt_leaf** is given. A branching run is a run for which **tt_leaf** is false and a leaf run a run for which **tt_leaf** is true. The decoder stops decoding the sequence of runs when the number of decoded leaf runs is superior to the number of branching runs. In the syntax a variable depth is used that counts the number of branching runs minus the number of leaf runs. This implies that the total number of runs is always an odd number.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



A	B	C

Figure 7-92 — Representing a simple polygon as a table of triangle runs.
A: depth-first order of traversal of the triangle tree starting from the root edge (green).
B: each triangle is classified by a 2-bit code as leaf(0), advance-right (1), advance-left (2), or branching (3).
C : The sequence of 2-bit codes is partitioned into runs ending in branching or leaf triangles. Each run is encoded as a (tt_run_length,tt_leaf) pair, plus a sequence of marching_edge bits.
Optionally, in the case of 3D mshes with polygonal faces, an additional sequence of polygon_edge bits is used to indicate which internal edges of the simple polygon correspond to original internal edges of the mesh, and which ones to virtual internal edges.

The **tt_leaf** flag is also used for the computation of the length of each run.

If the length of each run is correctly decoded and does not have virtual triangles, then the sum of the length of all runs must be equal to the length of the bounding loop constructed by the vertex graph decoder minus 2 (assuming that the bounding loop is correctly reconstructed).

However, if a run is partitioned, the triangle tree bitstream containes one or two virtual leaf triangles. Hence when decoding triangle data, the triangle data corresponding to the virtual leaf triangles shall not be decoded.

There are one or two virtual triangles if the number of triangles in the current partition is less than **codap_right_bloop_idx** - **codap_left_bloop_idx** - 1. Otherwise there are no virtual triangles. If there are virtual triangles and the third-last triangle is a branching triangle, the last two triangles in the partition are the virtual triangles. Other wise only the last triangle is virtual.

7.13.8 Triangle Data Decoder

The triangle data decoder operates in three steps. It first decodes a header, then a root triangle, which is followed by a sequence of triangles. Note that this sequence is empty if there is a single triangle in the current connected component.

The header is a boolean flag **triangulated** that indicates if all faces in the connected component are triangles. If this flag indicates true, the **polygon_edge** field is not coded.

The decoding of a root triangle and of a non-root triangle are the same with the following exception. The **polygon_edge** field is never coded for a root triangle of a connected component, and the number of property samples may differ.

Two fields indicate connectivity information for the current triangle: **marching_edge** and **polygon_edge**. **marching_edge** is not coded for branching and leaf triangles. **polygon_edge** is not coded for the root triangle or if the **triangulated** flag is set true. However in a partition of partition_type_2, the **polygon_edge** is coded for the root triangle if the **triangulated** field is set false

The number of per vertex property samples may differ depending on the boundary prediction mode and visited vertices. If the boundary prediction mode is restricted, then the two starting points are already visited in the

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

previous partition and only one root sample is coded. However, if the remaining vertex is also visited, there are no samples coded. If the boundary prediction mode is extended, the root triangle always has all 3 samples.

Table 7-24 — Number of samples to decode in a root triangle for each binding

Binding	# samples	Detail
None	0	
per vertex	0, 1, or 3	1 root and 2 non-root samples
per face	1	1 root sample
per corner	3	2 root and 1 non-root samples

For non-root triangles, the triangle data decoder decodes zero or one sample for each property bound per vertex. If the opposite vertex of the current triangle has been visited before, the value of the sample is already known and does not need to be decoded again. If the opposite vertex of the current triangle has never been visited before, then a sample is read. The bounding loop look-up table and the sizes of subtrees of the triangle tree are necessary to determine whether the opposite vertex of the current triangle has been visited previously. The boundary prediction mode is necessary to determine whether to consider the visited vertices within the partition or including every vertices visited in the previous partitions.

Table 7-25 — Number of samples to decode in a non-root triangle for each binding. The number of samples may be conditioned.

Binding	# samples	Condition
None	0	
per vertex	0 or 1	vertex never visited
per face	0 or 1	**polygon_edge** is set
per corner	1 or 3	**polygon_edge** is set

The traversal order is fixed in basic topological surgery. However, in order to change traversal order, **td_orientation** field is coded at a branching triangle on the main stem of the triangle tree in the partition. In other words, if the number of previously decoded branching triangles is equal to that of leaf triangles within the current partition, i.e. the depth value is zero, the **td_orientation** field is coded.

7.13.8.1 Decoding a root sample

A root sample is the first decoded sample in a root triangle. If there are several samples in a root triangle only the first sample is a root sample. The remaining samples in the root triangle are non-root samples.

The **coord_quant** bits of each coordinate of a root geometry sample are received starting with the most significant.

7.13.8.2 Decoding a non-root sample

A coordinate of a non-root geometry sample is composed of **coord_quant** bits, plus a sign bit. The coordinate is coded bit by bit starting from the most significant. The leading 0's, if any, and the leftmost 1, if any, are **coord_leading_bit**'s are coded using an adaptive probability estimation model. If not all **coord_leading_bit**s are 0's a **coord_sign_bit** is coded. Finally **coord_quant** minus the number of **coord_leading_bit**s **coord_trailing_bit**s are coded using a fixed probability model.

3***coord_quant** probability estimation models are defined to code the **coord_leading_bit** field. The probability model is selected according to the bit position and to the coordinate number.

The decoded value is not the actual value of the sample, but a correction that has to be applied to the prediction to obtain the reconstructed sample. The reconstructed sample is equal to the sum of the prediction and of the decoded value. The computation of the prediction is described below.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

It may happen that a reconstructed value is outside the bounding box, in which case a masking operation is carried out to insure that the final reconstructed value is indeed inside the bounding box. The masking operation depends on the quantisation step **coord_quant**. The **coord_quant** less significant bits of the reconstructed value are kept as is, and all other (more significant) bits are cleared.

The same structure applies to the other properties.

7.13.8.3 Property Prediction

For increased coding efficiency, the value of a property is coded relative to a prediction. This subclause describes the computation of the prediction for each type of binding. Table 7-26 and Table 7-27 describe all the valid combinations of property binding, property type, and prediction type.

Table 7-26 — Valid property bindings for each property type

binding	meaning	coord	normal	color	texCoord
00	not encoded		X	X	X
01	per vertex	X	X	X	X
10	per face		X	X	
11	per corner		X	X	X

Table 7-27 — Valid combinations of prediction type and property binding

pred_type	nlambda	binding
no_prediction	not coded	bound_per_vertex, bound_per_face, or bound_per_corner
tree_prediction	1	bound_per_face or bound_per_corner
parallelogram_prediction	3	bound_per_vertex

For tree and parallelogram prediction, the prediction of the properties in the current triangle are always based on the values in the previous triangle in the triangle tree. The prediction is weighted by the lambda coefficients decoded by the header decoder.

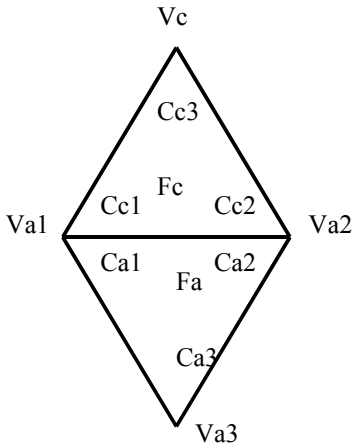


Figure 7-93 — Prediction of property values.
The value at the current vertex Vc is predicted from the values at ancestor vertices Va1, Va2 and Va3.
The value at the current face Fc is predicted from the value at ancestor face Fa.
The value at the current corner Cc1 is predicted from the value at ancestor corner Ca1.
The value at current corner Cc2 is predicted from the value at ancestor corner Ca2.
The value at current corner Cc3 is predicted from the value at current corner Cc1.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.13.8.3.1 Tree prediction

If the property is bound per face, the prediction of Fc is equal to Fa. If the current triangle is the root of the triangle tree, Fc is not predicted.

If the property is bound per corner, the prediction of Cc1 is equal to Ca1, the prediction of Cc2 to Ca2, and the prediction of Cc3 to Cc1. If the current triangle is the root of the triangle tree, Cc1 is not predicted, Cc2 is predicted by Cc1, and Cc3 by Cc1.

7.13.8.3.2 Parallelogram prediction

If the property is bound per vertex, the prediction of Vc is equal to lambda1*Va1 + lambda2*Va2 + lambda3*Va3. If the current triangle is the root of the triangle tree, Va1 and Va2 must be encoded too, Va1 is not predicted, Va2 is predicted by Va1, and Vc by Va1.

7.13.8.4 Inverse quantisation

There are two inverse quantisation procedures. The first one applies to geometry, colors and texture coordinates. The second applies to normals.

Given a quantised geometry sample (qx, qy, qz), the reconstructed geometry sample (x,y,z) is obtained as:

x = **coord_xmin** + ((1<<**coord_quant**)-1)*__coord_size__*xq
y = **coord_ymin** + ((1<<**coord_quant**)-1)*__coord_size__*yq
z = **coord_zmin** + ((1<<**coord_quant**)-1)*__coord_size__*zq

The procedure for colors and texture coordinates is the same.

There is a special inverse quantisation procedure for normals. Normals are always quantised to an odd number of bits **normal_quant**. The 3 most significant bits of **normal_quant** indicate the octant the normal lies in, and the remaining bits define an index within the octant.

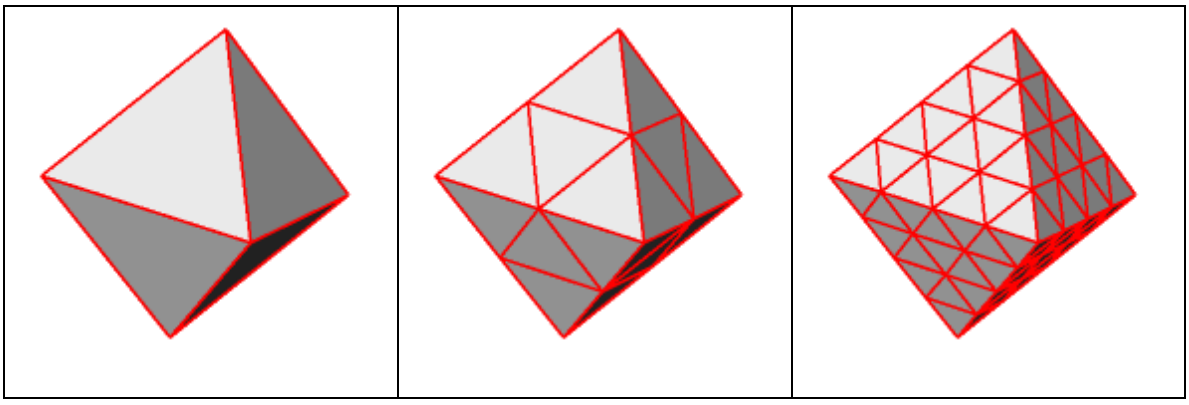


Figure 7-94 — The quantisation of normals is based on the hierarchical subdivision of the unit octahedron. Left: unit octahedron. Middle: unit octahedron after one subdivsion. Right: unit octahedron after two subdivisions.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

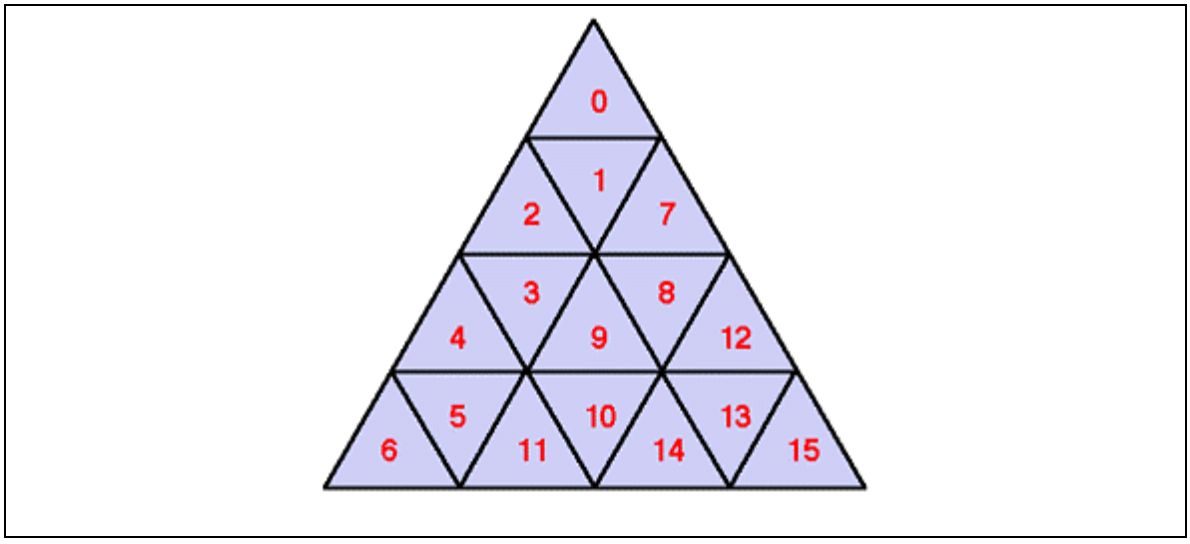


**Figure 7-95 — Numbering of the triangles within an octant.
Note that the top corner corresponds to |z|=1,
the leftmost corner to |x|=1 and the rightmost corner to |y|=1.**

Given an index i, and the 3 sign bits sx, sy and sz, the normal (x,y,z) can be analytically reconstructed as follows:

```
nbins = 1 << ((normal_quant - 3) / 2);

y0 = nbins - ceil(sqrt(nbins*nbins - i));
x0 = i + y0*y0;
skew = (x0 & 1)*2.0/3.0;
x0 = (x0 >> 1) & (nbins - 1);

x = (float)x0 + skew;
y = (float)y0 + skew;
z = (float)nbins - x - y;

n = 1.0/sqrt(x*x + y*y + z*z);
x = (sx) ? -x*n : x*n;
y = (sy) ? -y*n : y*n;
z = (sz) ? -z*n : z*n;
```

7.13.9 Forest Split decoder

In the progressive forest split (PFS) scheme, a 3D model is represented as a low resolution mesh followed by a sequence of forest split operations. A forest split operation is represented by a forest in the graph of vertices and edges of a mesh, a sequence of simple polygons, and a sequence of vertex coordinates and property updates. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with one of the simple polygons, and finally, displacing the new vertices and properties. An optional pre-smoothing step can be used as a global predictor for the vertex coordinates, after the and properties of the new faces bound per face and per corner are updated. Also, and optional post-smoothing step can be used to reduce the quantisation artifacts after all the updates have been applied.

7.13.9.1 Enumeration of Mesh Elements

Given a manifold triangular 3D mesh with *V* vertices, *T* triangles and no boundaries, we assume that the vertices have consecutive *vertex indices* in the range *0*, …, *V-1*, and the triangles have consecutive *triangle indices* in the range *0,…,T-1*. The edges of the mesh, which are represented by pairs of vertex indices (*i,j*), with *i* < *j*, are ordered lexicographically and assigned consecutive *edge indices* in the range *0,…,E-1*. The trees in the forest are ordered according to the minimum vertex index of each tree. The *root vertex* of each tree in the forest is the leaf of the tree with the minimum index. Starting at the root, the boundary loop created by cutting along the tree can be traversed

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

in cyclic fashion in one of the two directions. The *root edge* of the tree is the only edge of the tree that has the root vertex as an endpoint. Of the two triangles incident to the root edge of the tree, the *root triangle* of the tree is the one with minimum triangle index. Of the two edges of the tree boundary loop corresponding to the root edge of the tree, the *root edge* of the tree boundary loop is the one incident to the root triangle.

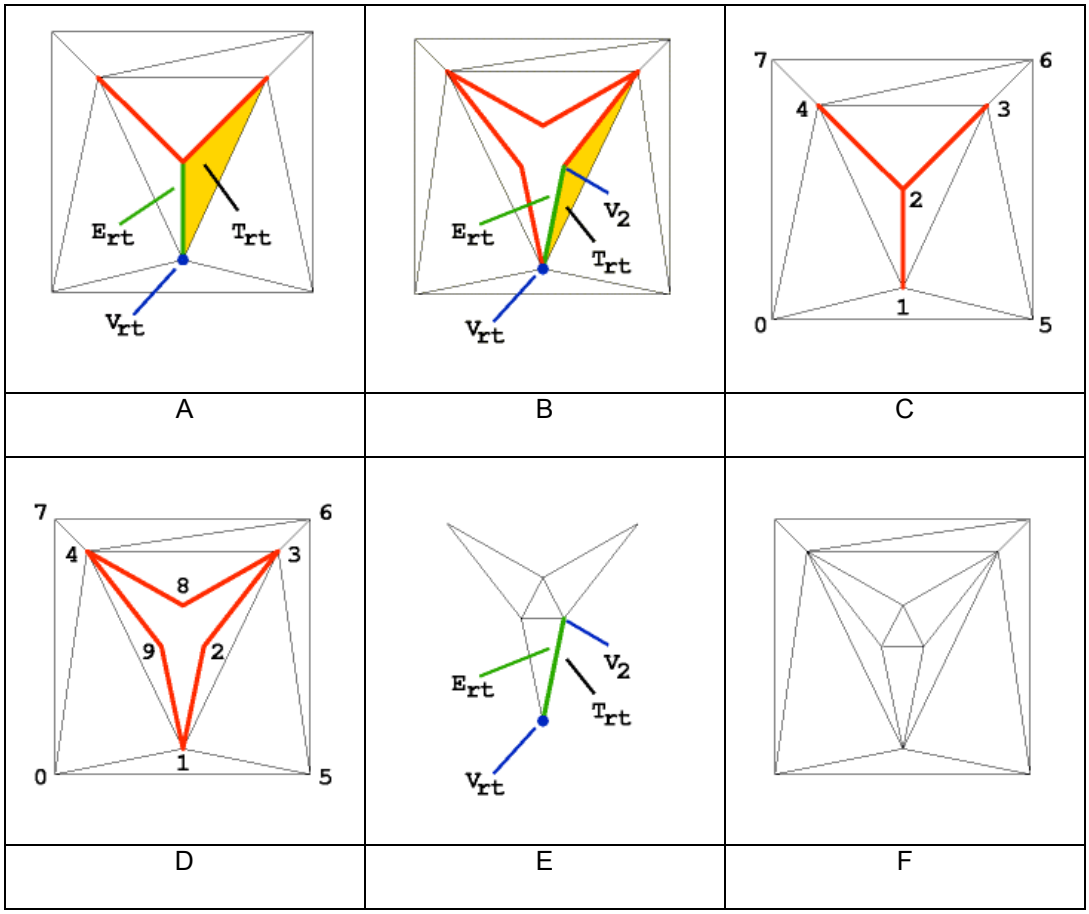


Figure 7-96 — When a mesh is cut through a tree of edges (red and green edges in A), a tree boundary loop (red and green edges in B) is created with each edge of the tree corresponding to two edges of the boundary loop. Some vertex indices are assigned before cutting (in C) to new tree boundary loop vertices, others are assigned subsequent indices (in D). The hole created by the cutting operation is filled by triangulating the boundary loop using a simple polygon (E) resulting in a refined mesh (F) with the same topological type as the initial mesh.

7.13.9.2 Decoding the forest

Figure 7-97 describes the algorithm to decode the forest of edges from the bitstream. Initially, the forest is empty. As edge marks are decoded, the corresponding marked edges are included in the forest. Edges which if marked would create loops are skiped, because they can be predicted as not being marked. The order of the edge bits in the bitstream correspond to the enumeration of edges defined above.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

```
// forest decoder
// marked edges form a forest in the graph of the mesh

int nV = number of vertices of mesh
Partition p(nV)
// p defines a partition of a set of nV elements
// initially each part contains a single element
// the method find(i) retrieves the number of the part containing element i
// the method join(i,j) joins the parts containing elements i and j, respectively

for ( every edge e={i,j} of mesh ) {
  bool skip_edge = false;
  bool edge_bit  = false;
  if(p.find (i)!=p.find(j)) {
    edge_bit = decode_bit();
  } else {
    // forest => edge_bit==false
    skip_edge = true;
  }
  e.isInForest = edge_bit;
  if(edge_bit==true)
    p.join(I,j);
}
```

Figure 7-97 — forest decoder pseudo-code

7.13.9.3 Cutting Through Forest Edges

Cutting through a forest of edges can be performed sequentially, cutting through one tree at a time. Each cut is typically a local operation, affecting only the triangles incident to vertices and edges of the tree. However, as in the Topological Surgery method, a single cut could involve all the triangles of the mesh. Cutting requires duplicating some tree vertices, assigning additional indices to the new vertices and fixing the specification of the affected triangles. If no tree vertex is a boundary vertex of the mesh, then the tree is completely surrounded by triangles. This is illustrated Figure 7-98 A,B.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

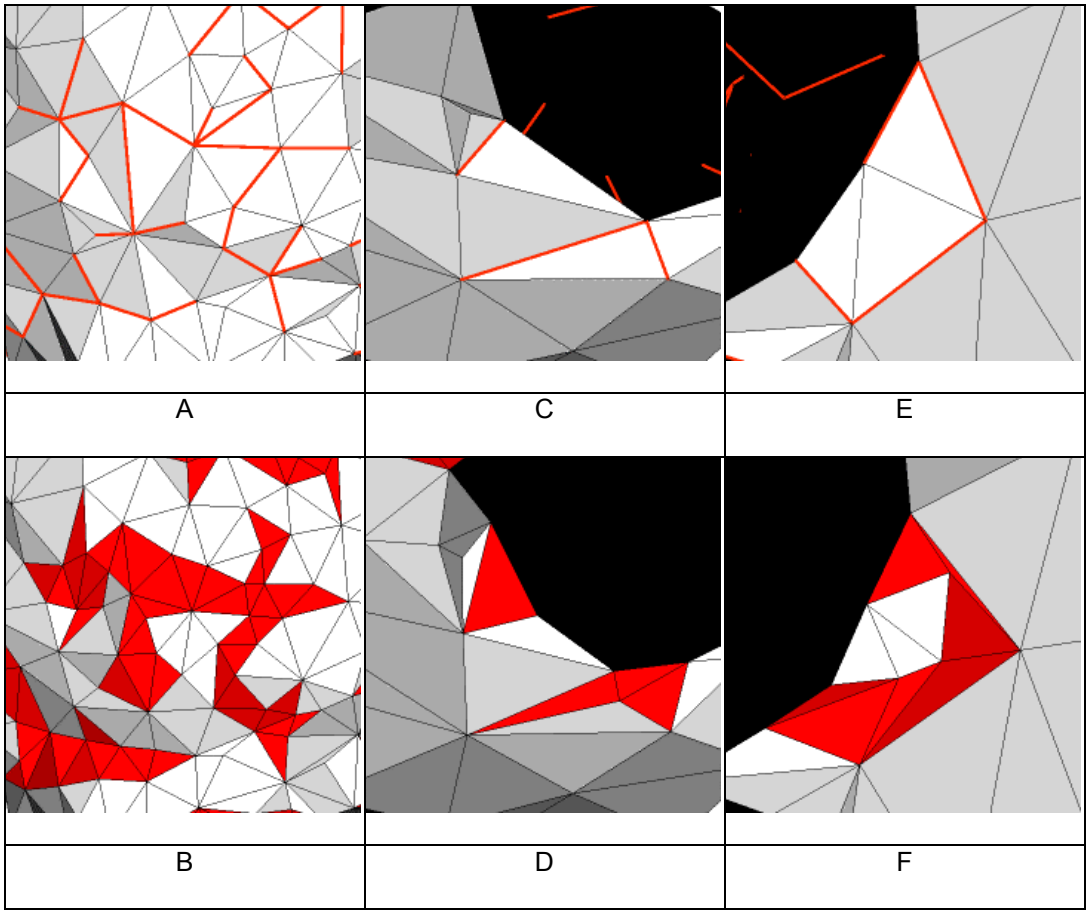


Figure 7-98 — Construction and triangulation of tree boundary loops.
A,B: No tree vertices belong to the mesh boundary.
C,D: A tree vertex isolated in the mesh boundary requires an extra tree loop edge.
E,F: A tree edge on the mesh boundary does not require an extra tree loop edge, but some of the new vertex indices may only be used by new triangles.
Note that the tree may have several contacts with the mesh boundary.

Starting at the root triangle, all the corners of affected triangles can be visited in the order of traversal of the tree boundary loop, by jumping from triangle to neighboring triangle, while always keeping contact with the tree. The direction of traversal is defined by going from the root edge e_{rt} towards the root triangle t_{rt}. This process produces a list of triangle corners, called the *corner loop*, whose values need to be updated with the new vertex indices. While traversing this list, we encounter runs of corners corresponding to the same vertex index before the cut. A new vertex index must be assigned to each one of these runs. To prevent gaps in the list of vertex indices we first need to reuse the vertex indices of the tree vertices, which otherwise would not be corner values of any triangles. The first visited run corresponding to one of these vertices is assigned that vertex index. The next visited run corresponding to the same vertex index is assigned the first vertex index not yet assigned above the number of vertices of the mesh before the cut. This procedure performs the topological cut. For example, in Figure 7-98 C, the vertex index values of the corners in the corner loop list are:

[12333332444444421111]

The list can be decomposed into 6 runs [11111], [2], [33333], [2], [444444] and [2]. As shown in Figure 7-98 D, the vertex indices that are assigned to these runs are 1,2,3,8,4,9. A tree with m edges containing no mesh boundary vertices creates a tree boundary loop of $2m$ edges. When one or more vertices belong to the mesh boundary, the boundary loop has some edges without incident triangles. Cases when vertices belong to the mesh boundary are illustrated in Figure 7-98 C,D, E,F. There are three cases: tree edges belonging to the boundary (Case 1, shown in Figure 7-98 E,F), a tree vertex that is isolated on the mesh boundary (Case 2, shown in Figure 7-98 C,D) or an isolated tree vertex on the boundary that happens to be the root vertex v_{rt} (Case 3, which is a sub-case of Case 2). Case 1 is treated similarly to the general case: we loop around the vertex tree, creating 2 new loop edges for each

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

marked edge. Some of these edges do not belong to any existing triangle. The method that is used for looping around boundary edges must take into account that there are no triangles on the other side. One possibility is to create fake faces on the other side of the boundary and keep the same method for jumping from triangle to triangle. Another possibility, which is exploited in the sample implementation, is to start jumping in the other direction when we hit the boundary. Except from these difference in the algorithm, the same method is used to assign new vertex indices for some of the loop corners. In Case 2, an extra edge should be added to the loop. In Case 3, an extra edge is added as well, but it remains to determine which of the two endpoints of the created edge will be the root vertex of the loop. By convention, it will be the endpoint to which the root triangle is incident. These three cases are illustrated in Figure 7-99. In summary, a tree with *m* edges containing *n* isolated boundary vertices creates a tree boundary loop of *2m+n* edges

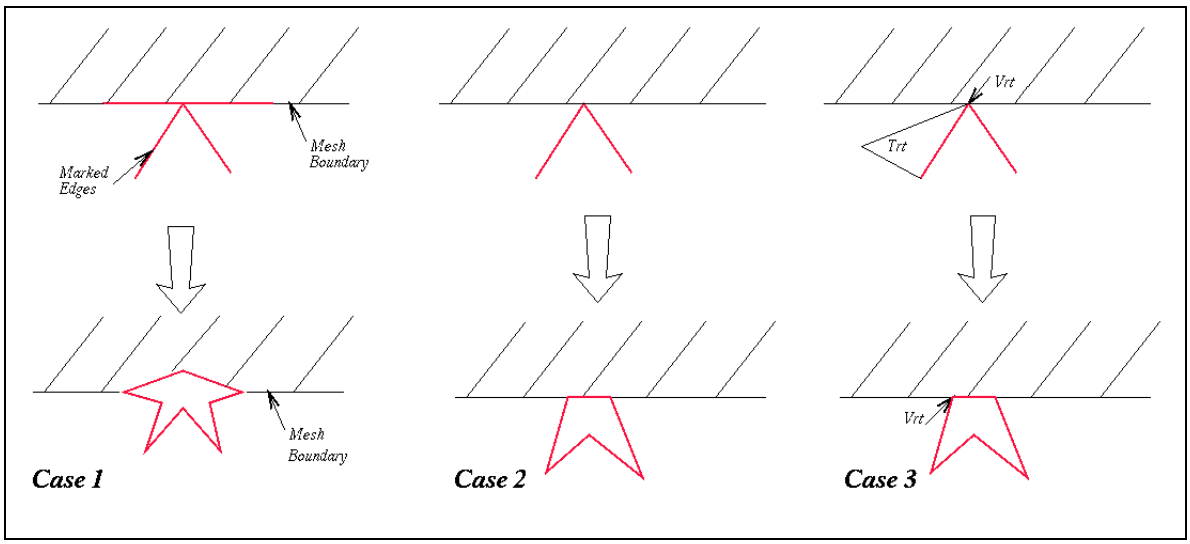


Figure 7-99 — Special cases encountered while constructing the tree boundary loop corrersponding to a tree touching the boundary of the mesh

7.13.9.4 Triangulating Tree Boundary Loops

By replacing each run of corners in the corner loop with the assigned vertex index, we construct a new list representing the tree boundary loop. If the tree boundary loop has m vertices, so does the corresponding polygon boundary loop. Each triangle *t = {i,j,k}* of the simple polygon defines a new triangle of the refined mesh by replacing the polygon boundary loop indices *i,j,k* with their corresponding tree boundary loop indices. Figure 7-98 E,F illustrates this. This is done using the list representing the tree boundary loop as a lookup table. The triangles of the simple polygon are visited in the order of a depth first traversal of its dual tree. The traversal starts with the triangle opposite to the root triangle and always traverses the left bran ch of a branching triangle first.

7.13.9.5 Simple polygon properties bound per face and per corner

The properties bound per face and per corners to the faces of the simple polygons used to fill the tree boundary loops are encoded as in the case of a connected component of a single resolution 3D mesh, predicting per face values along the face tree, and per corner values along the derived corner tree.

7.13.9.6 Tree loop vertex, face, and corner properties

Vertex coordinates and properties bound per vertex corresponding to new tree loop vertices are first assigned the same values as the corresponding tree vertex coordinates and properties before the cut. To prevent the appearance of holes, these vertices are displaced after the boundary loops are triangulated. Rather than encoding the new absolute positions of the tree loop vertices and corresponding per vertex properties, we encode their displacement with respect to the position before the forest split operation, and optionally after pre-smoothing (pre-smoothing is only applied to vertex coordinates). These values are encoded in the order of traversal of the tree loop vertices.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Properties bound per face and per corner to tree loop faces and corners are encoded in a similar fashion, except that properties bound per face (a face may be a tree loop face for two or more trees) are encoded with no repetition, skipping values associated with faces previously visited during the current update cycle.

7.13.9.6.1 Pre and post-smoothing

The differences between vertex positions before and after each forest split operation can be made smaller by representing these errors as the sum of a global predictor plus a correction. We use the signal processing smoothing method [4] in Annex O as a global predictor. Figure 7-100 shows a pseudo-code implementation of the scheme used for both pre and post smoothing.

```
// pre and post smoothing

int    nV     // number of vertices of mesh
Vertex v[nV]  // array of vertices of the mesh
Vertex dv[nV] // auxiliary array
int    nv[nV] // auxiliary array

// determine sharp vertices from sharp edges
for(i=0;i<nV;i++)
  v[i].isSharp = false
for ( every edge e={i,j} of mesh ) {
  if(e.isSharp) {
    v[i].isSharp = true
    v[j].isSharp = true
  }
}

// main loop
for(n=0;n<n_smooth;n++) {

// accumulate displacements
  dv = 0;
  nv = 0;
  for ( every edge e={i,j} of mesh ) {
    dve = v[j] - v[i];
    if(v[i].isFixed==false && (e.isSharp==true || v[i].isSharp==false)) {
      dv[i] += dve;
      nv[i] += 1;
    }
    if(v[j].isFixed==false && (e.isSharp==true || v[j].isSharp==false)) {
      dv[j] -= dve;
      nv[j] += 1;
```


--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

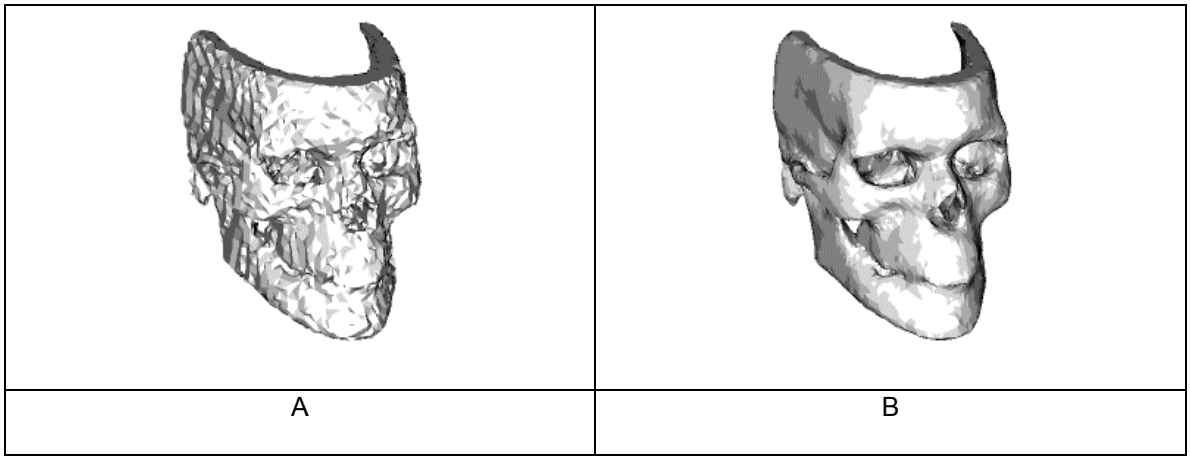
```
  }
 }

 // normalize displacements and apply
 float a = (n is even)?lambda_smooth:mu_smooth;
 for(i=0;i<nV;i++)
  if(nv[i]>0) {
     dv[i] /= nv[i]
     v[i] += a*dv[i];
   }
}
```

Figure 7-100 — Pseudo-code for smoothing operator with discontinuty edges and fixed vertices

An optional pre-smoothing step is applied as a global predictor. After the connectivity refinement step of a forest split operation is applied, the new vertices are positioned where their corresponding vertices in the previous level were positioned and the mesh has many edges of zero length (all the new triangles have zero surface area). The signal processing smoothing method, which tends to equalize the size of neighboring triangles, brings the endpoints of most such edges apart, most often reducing the distance to the desired vertex positions. The corrections, the differences between the vertex positions after the split operation and the result of smoothing the positions before the split operation, are then quantised.

If the 3D mesh approximates a smooth shape, a post-smoothing step can be applied to reduce the visual artifacts produced by the quantisation process. In many cases, even fewer bits per coordinate can be used in the quantisation process without a significant perceptual difference, also reducing the total length of the encoded vertex displacements.

In both the pre and post-smoothing steps optional discontinuity edges and/or fixed vertices can be specified in the bitstream, together with the smoothing parameters. If so, the same discontinuity edges and/or fixed vertices are used for both steps.



A	B

Figure 7-101 — Smoothing a 3D model.
(A) Original 3D model. (B) 3D model after applying the smoothing operator introduced in [4] in Annex O.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.13.10 Arithmetic decoder

This section describes the QF-coder for arithmetic decoding. C++-style routines are provided. The arithmetic decoder relies on a set of variables that are described in the table below.

Register	Description
A	size of current interval
C	current arithmetic code
count	number of renormalisations before reading next byte
symbol	value of decoded symbol
Qe	probability estimate for least probable symbol
MPS	value of most probable symbol

The registers Qe and MPS are dependent upon a context. For each context, there is a state variable including the Qe and MPS registers.

7.13.10.1 Initialisation

The arithmetic coder reads data from the input on a per byte basis. Therefore the stream pointer shall be byte aligned before initialising the arithmetic decoder.

In the initialisation process, all variables are initialised to proper values. 2 bytes are read from the input stream into register C.

```
void qf_start() {

  A = 0x00010000;

  C = 0x00010000;

  count = 0;

  sw_renorm_decode();

}
```

7.13.10.2 Decoding a symbol

When a binary symbol is decoded, the interval is split into two. The half in which C lies determines the value of the bit. The interval is then reduced to the corresponding half. If the size of the new interval is smaller than 0x80000000, then the interval is renormalised and the probability estimator updated.

```
void qf_decode(int *symbol, QState *state) {

  *symbol = state->MPS;


  A -= state->Qe;
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  if (C <= A) {
    if (A & 0x80000000)
      return;
    *symbol ^= A < state->Qe;
  }
  else {
    *symbol ^= A >= state->Qe;
    C = C - A;
    A = state->Qe;
  }
  state->update(*symbol);
  renorm_decode();
}
```

7.13.10.3 Renormalisation

In the renormalisation step, the size of the interval is doubled until it reaches at least 0x80000000. Each time the size of the interval is doubled a bit is consumed. When no more bit is available for consumption in the less significant bits of the register C, a new byte is retrieved from the stream.

```
void qf_renorm() {
  do {
    if (count == 0) {
      int b = get_byte();
      C -= 0xff << 8;
      C += b << 8;
       count = (b == 0) ? 7 : 8;
    }
    count--;
    C = C << 1;
    A = A << 1;
  } while (!(A & 0x80000000));
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.13.10.4 Probability estimation

A majority of the fields that are arithmetic coded benefit from adaptive probability estimation. This adaptive estimation is defined by a Markov Model. Each state of the model defines a probability of the LPS, a next state if the currently coded symbol is the MPS, a next state if the currently coded symbol is the LPS, and a flag that indicates if the value of the MPS should be changed if the currently coded symbol is the LPS. The Markov model to be used is defined in Figure 7-84. It is named FA-JPEG for Fast-Attack JPEG, and has been designed such as to minimize the number of states.

A fixed probability of the LPS and a fixed MPS are defined for the fields for which there is no adaptive probability estimation.

Table 7-28 lists the probability estimation used for each of the fields.

The models are global and initialised to index = 0, MPS = 0 before the first connected component.

The context of order 1 models is reset before each component.

The state is updated every time the renormalisation procedure is called.

Table 7-28 — List of fields that are arithmetic coded and their associated probability estimation

Field	probability estimation	context
last_component	FA-JPEG	last_component_context
has_stitches	FA-JPEG	has_stitches_context
codap_last_vg	FA-JPEG	codap_last_vg_context
codap_vg_id	FA-JPEG	zero_context
codap_left_bloop_idx	FA-JPEG	zero_context
codap_right_bloop_idx	FA-JPEG	zero_context
codap_bdry_pred	FA-JPEG	zero_context
vg_simple	FA-JPEG	vg_simple_context
vg_last	FA-JPEG	vg_last_context
vg_forward_run	FA-JPEG	vg_forward_run_context
vg_loop_index	FA-JPEG	vg_loop_index_context
vg_run_length	FA-JPEG	vg_run_length_context
vg_leaf	FA-JPEG	vg_leaf_context
vg_loop	FA-JPEG	vg_loop_context
tt_run_length	FA-JPEG	tt_run_length_context
tt_leaf	FA-JPEG	tt_leaf_context
codap_branch_len	FA-JPEG	codap_branch_len_context
triangulated	FA-JPEG	triangulated_context
marching_edge	FA-JPEG, order 1, initial context = 1	marching_edge_context[0..1]
td_orientation	FA-JPEG	td_orientation_context
polygon_edge	FA-JPEG, order 1, initial context = 1	polygon_edge_context[0..1]
coord_bit	fixed Qe = 0x5601, MPS = 0	zero_context
coord_leading_bit	FA-JPEG, 3*coord_quant contexts	coord_leading_bit_context[0..3*coord_quant-1]
coord_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context
coord_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
normal_bit	fixed Qe = 0x5601, MPS = 0	zero_context
normal_leading_bit	FA-JPEG, normal_quant contexts	normal_leading_bit_context[0..normal_quant-1]
normal_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

normal_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
color_bit	fixed Qe = 0x5601, MPS = 0	zero_context
color_leading_bit	FA-JPEG, 3*color_quant contexts	color_leading_bit_context[0..3*color_quant-1]
color_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context
color_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
texCoord_bit	fixed Qe = 0x5601, MPS = 0	zero_context
texCoord_leading_bit	FA-JPEG, 2*texCoord_quant contexts	textCoord_leading_bit_context[0..2*texCoord_quant-1]
texCoord_sign_bit	fixed Qe = 0x5601, MPS = 0	zero_context
texCoord_trailing_bit	fixed Qe = 0x5601, MPS = 0	zero_context
stitch_cmd	FA-JPEG	stitch_cmd_context
stitch_pop_or_get	FA-JPEG	stitch_pop_or_get_context
stitch_pop	FA-JPEG	stitch_pop_context
stitch_stack_index	FA-JPEG	stitch_stack_index_context
stitch_incr_length	FA-JPEG	stitch_incr_length_context
stitch_incr_length_sgn	FA-JPEG	stitch_incr_length_sgn_context
stitch_push	FA-JPEG	stitch_push_context
stitch_reverse	FA-JPEG	stitch_reverse_context
stitch_length	FA-JPEG	stitch_length_context
pfs_forest_edge	FA-JPEG	pfs_forest_edge_context
smooth_with_sharp_edges	fixed Qe = 0x5601, MPS = 0	zero_context
smooth_sharp_edge	FA-JPEG	smooth_sharp_edge_context
smooth_with_fixed_vertices	fixed Qe = 0x5601, MPS = 0	zero_context
smooth_fixed_vertex	FA-JPEG	smooth_fixed_vertex_context
other_update	fixed Qe = 0x5601, MPS = 0	zero_context

Table 7-29 — The Fast Attack JPEG (FA-JPEG) Markov model to estimate probabilities. For each state, we define the probability of the LPS, the next state in the event of an MPS, the next state in the event of an LPS, and a switch MPS flag in the event of an LPS.

State	Probability of LPS	Next state if MPS	Next state if LPS	Switch MPS if LPS
0	0x5601	1	1	yes
1	0x3401	2	6	no
2	0x1801	3	9	no
3	0x0ac1	4	12	no
4	0x0521	5	29	no
5	0x0221	38	33	no
6	0x5601	7	6	yes
7	0x5401	8	14	no
8	0x4801	9	14	no
9	0x3801	10	14	no
10	0x3001	11	17	no
11	0x2401	12	18	no
12	0x1c01	13	20	no
13	0x1601	29	21	no
14	0x5601	15	14	yes

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15	0x5401	16	14	no
16	0x5101	17	15	no
17	0x4801	18	16	no
18	0x3801	19	17	no
19	0x3401	20	18	no
20	0x3001	21	19	no
21	0x2801	22	19	no
22	0x2401	23	20	no
23	0x2201	24	21	no
24	0x1c01	25	22	no
25	0x1801	26	23	no
26	0x1601	27	24	no
27	0x1401	28	25	no
28	0x1201	29	26	no
29	0x1101	30	27	no
30	0x0ac1	31	28	no
31	0x09c1	32	29	no
32	0x08a1	33	30	no
33	0x0521	34	31	no
34	0x0441	35	32	no
35	0x02a1	36	33	no
36	0x0221	37	34	no
37	0x0141	38	35	no
38	0x0111	39	36	no
39	0x0085	40	37	no
40	0x0049	41	38	no
41	0x0025	42	39	no
42	0x0015	43	40	no
43	0x0009	44	41	no
44	0x0005	45	42	no
45	0x0001	45	43	no

7.14 NEWPRED mode decoding

NEWPRED is a technique in which the reference picture for inter-frame coding is replaced adaptively. This may be according to the upstream messages from the decoder.

7.14.1 Decoder Definition

In NEWPRED mode, decoder may have the additional reference VOP memory to store previously decoded VOPs. When the reference VOP is indicated by vop_id_for_prediction, the decoder select reference VOP from reference VOP memory or additional reference VOP memory to switch the reference picture to the indicated one.

The unit of switching reference VOP, which is NEWPRED segment (NP segment), is indicated by newpred_segment_type. When this flag is “Video Packet”, reference VOP shall be switched by Video Packets. In this case, after switching reference VOP, outside of current Video Packet shall be padded in the same way as Unrestricted MC. Each Video Packet boundaries are treated as VOP boundaries, including the treatment of motion vectors which cross those boundaries. The unrestricted motion vector technique is used for Video Packet boundaries. MV prediction and Intra AC/DC prediction are restricted to prediction within the Video Packet. When newpred_segment_type flag is “VOP”, reference VOP shall be switched by VOPs.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Regardless of newpred_segment_type, the decoder shall not refer to any NP segment which is decoded before the last I_VOP.

7.14.1.1 Position of resynchronization marker

When newpred_segment_type is “Video Packet”, it is implied that the positions of Resynchronization Markers are not changed between I_VOPs.

7.14.2 Upstream message

The decoder may return the upstream message to the encoder based on the upstream syntax defined in 6.2.12. Which type of message is required for the encoder is indicated in requested_upstream_message_type in the VOL header of the downstream data. To return the upstream message is not normative procedure.

7.15 Output of the decoding process

This subclause describes the output of the theoretical model of the decoding process that decodes bitstreams conforming to this part of ISO/IEC 14496.

The visual decoding process input is one or more coded visual bitstreams (one for each of the layers). The visual layers are generally multiplexed by the means of a system stream that also contains timing information.

7.15.1 Video data

The output of the video decoding process is a series of VOPs that are normally the input of a display process. he order in which fields or VOPs are output by the decoding process is called the display order, and may be different from the coded order (when B-VOPs are used).

7.15.2 2D Mesh data

The output of the decoding process is a series of one or more mesh object planes. The mesh object planes are normally input to a compositor that maps the texture of a related video object or still texture object onto each mesh. The coded order and the composition order of the mesh object planes are identical.

7.15.3 Face animation parameter data

The output of the decoding process is a sequence of facial animation parameters. They are input to a display process that uses the parameters to animate a face object.

7.16 Video object decoding for the studio profile

This subclause specifies the video object decoding for the studio profile.

7.16.1 Video decoding process

(The identical description to subclause 7.1)

7.16.2 Higher syntactic structures

The various parameters and flags in the bitstream for StudioVideoObjectLayer(), Group_of_StudioVideoObjectPlane(), StudioVideoObjectPlane(), StudioSlice(), StudioMacroblock(), StudioBlock() and StudioDPCMBlock(), as well as other syntactic structures related to them shall be interpreted as discussed earlier. Many of these parameters and flags affect the decoding process. Once all the macroblocks in a given VOP have been processed, the entire VOP will have been reconstructed.

Texture data in a macroblock shall have been encoded by either DCT or DPCM.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If compression_mode == 1, texture data in the macroblock shall have been encoded by DCT. The decoding process is indicated in subclause 7.16.4.

If compression_mode == 0, texture data in the macroblock shall have been encoded by DPCM. The decoding process is indicated in subclause 7.16.5.

Reconstructed fields shall be associated together in pairs to form reconstructed frames. (See “vop_structure” in 6.3.13.5.)

The sequence of reconstructed frames shall be reordered as described in 6.3.13.5.

If progressive_sequence == 1 the reconstructed frames shall be output from the decoding process at regular intervals of the frame period as shown in Figure 7-117.

If progressive_sequence == 0 the reconstructed frames shall be broken into a sequence of fields which shall be output from the decoding process at regular intervals of the field period as shown in Figure 7-118. In the case that a frame picture has repeat_first_field == 1 the first field of the frame shall be repeated after the second field. (See “repeat_first_field” in 6.3.13.5.)

7.16.3 VOP reconstruction

The luminance and chrominance values of a VOP from the decoded texture and motion information are reconstructed as indicated in this subclause. Figure 7-102 represents the process.

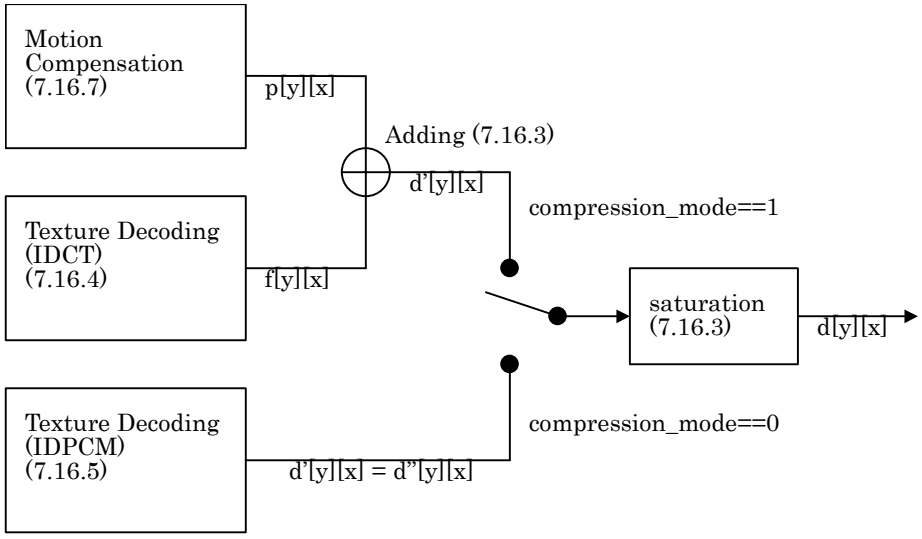


Figure 7-102 — VOP reconstruction from the decoded texutre and motion compensated data

1. In case of INTRA macroblocks of compression_mode==1, the luminance and chrominance values f[y][x] from the decoded texture data form the luminance and chrominance values: d’[y][x] = f[y][x].

2. In case of INTER macroblocks, first the prediction values p[y][x] are calculated using the decoded motion vector information and the texture information of the respective reference VOPs. Then, the decoded texture data f[y][x] is added to the prediction values, resulting in the luminance and chrominance values: d’[y][x] = p[y][x] + f[y][x].

3. In case of INTRA macroblocks of compression_mode==0, the luminance and chrominance values d’’[y][x] from the decoded texture data form the luminance and chrominance values: d’[y][x] = d’’[y][x].

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4. Finally, the calculated luminance and chrominance values are saturated so that

$$d[y][x] = \begin{cases} 2^{bits_per_pixel} - 1; & d'[y][x] > 2^{bits_per_pixel} - 1 \\ d'[y][x]; & 0 \le d'[y][x] \le 2^{bits_per_pixel} - 1 \\ 0; & d'[y][x] < 0 \end{cases}$$

NOTE The saturation defined above limits output sample values for Y, Cr and Cb to the range [0:$2^{bits_per_pixel}$-1]. Therefore, the values which are assigned as the reserved code words for timing reference in ITU-R BT.601 and BT.709 can occasionally occur.

7.16.4 Texture decoding from DCT coefficients

This subclause describes the process used to decode the texture information of a VOP in case of compression_mode==1. The process of video texture decoding from coded DCT coefficients is given in Figure 7-103.

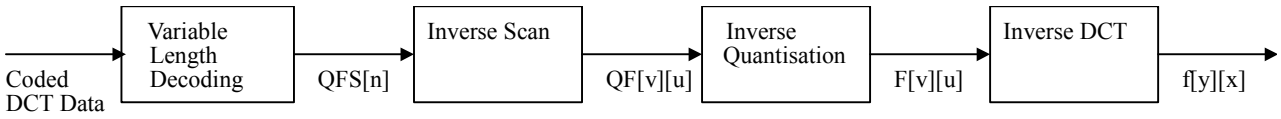


Figure 7-103 — Video Texture Decoding Process from DCT coefficients

7.16.4.1 Variable length decoding

This subclause explains the decoding process. Subclause 7.16.4.1.1 specifies the process used for the DC coefficients (n=0) in an intra coded block. (n is the index of the coefficient in the appropriate zigzag scan order). Subclause 7.16.4.1.2 specifies the decoding process for all other coefficients; AC coefficients ($n \neq 0$) and DC coefficients in non-intra coded blocks.

Let *cc* denote the colour component. It is related to the block number as specified in Table 7-30. In case of rgb_components==0, *cc* is zero for the Y component, one for the C_b component and two for the C_r component. In case of rgb_components==1, zero for the G component, one for the B component and two for the R component.

Table 7-30 — Definition of *cc*, colour component index

	cc		
Block Number	**4:2:0**	**4:2:2**	**4:4:4**
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	1	1	1
5	2	2	2
6		1	1
7		2	2
8			1
9			2
10			1
11			2

--`,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.4.1.1 DC coefficients decoding in intra blocks

Differential DC coefficients in blocks in intra macroblocks are decoded as a variable length code denoting dct_dc_size as defined in Table B.44 and Table B.45 in Annex B, and a fixed length code dct_dc_differential (Table B.46). The dct_dc_size categorizes the DC coefficients according to their "size". For each category additional bits are appended to the dct_dc_size code to uniquely identify which difference in that category actually occurred (Table B.46). This is done by appending a fixed length code, dct_dc_differential, of dct_dc_size bits.

In case of rgb_components==0, if *cc* is zero then Table B.44 shall be used for dct_dc_size. If *cc* is non-zero then Table B.45 shall be used for dct_dc_size.

In case of rgb_components==1, Table B.44 shall be used for dct_dct_size for all of the components, *cc*.

Three predictors are maintained, one for each of the colour components, *cc*. Each time a DC coefficient in a block in an intra macroblock is decoded the predictor is added to the differential to recover the actual coefficient. Then the predictor shall be set to the value of the coefficient just decoded. At various times, as described below, the predictors shall be reset. The reset value except when a macroblock is encoded as DPCM residuals is derived from the combination of the parameters bits_per_pixel , dct_precision and intra_dc_precision as;

2 ^ (bits_per_pixel+dct_precision+intra_dc_precision-1)

The predictors shall be reset to the above reset value at the following times:

- Whenever a macroblock is decoded if intra_predictors_reset == 1.
- At the start of a slice.
- Whenever a non-intra macroblock is decoded.
- Whenever a macroblock is skipped. i.e. not_coded == 1.
- Whenever a macroblock and a block is transparent.

When a macroblock is encoded as DPCM residuals, the reset value is derived as;

block_mean x (2 ^ (dct_precision+intra_dc_precision))

Because block_mean is encoded for each component independently, the reset value for each component can be different.

The predictors are denoted *dct_dc_pred*[*cc*].

QFS[0] shall be calculated from dct_dc_size and dct_dc_differential by any process equivalent to:

```
if ( dct_dc_size == 0 ) {
        dct_diff = 0;
} else {
        half_range = 2 ^ ( dct_dc_size - 1 );
        if ( dct_dc_differential >= half_range )
            dct_diff = dct_dc_differential;
        else
            dct_diff = (dct_dc_differential + 1) - (2 * half_range);
}
QFS[0] = dct_dc_pred[cc] + dct_diff;
dct_dc_pred[cc] = QFS[0]
```

Annex NA NOTE 1 - The symbol ^ *denotes power (not XOR)*.

Annex NB NOTE 2 - *dct_diff* and *half_range* are temporary variables which are not used elsewhere in this specification.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

It is a requirement of the bitstream that QFS[0] shall lie in the range:

— 0 to ((2^(dct_precision + bits_per_pixel + intra_dc_precision))-1)

7.16.4.1.2 Other coefficients

All coefficients with the exception of the DC intra coefficients shall be decoded using a set of 12 VLC tables, T[0] to T[11] . The default set of the 12 tables is defined in Table B.50 to Table B.61 for intra macroblocks and Table B.62 to Table B.73 for inter macroblocks. The set of tables can be downloaded using the extension syntax of vlc_code_extension().

The decision rule to select one table from the 12 tables for decoding a symbol is defined in this subclause.

In all cases a variable length code shall first be decoded using one of the tables. The decoded value of this code denotes one of three courses of action:

1) End of Block. In this case there are no more coefficients in the block in which case the remainder of the coefficients in the block (those for which no value has yet been decoded) shall be set to zero. This is denoted by “End of block” in the syntax specification of 6.2.13.9.

2) A “normal” coefficient. In which a symbol denoting a value of *run* and/or *level* is decoded followed by a fixed length code of the size indicated by the variable length code according to Table 7-31. The variable length code categorizes symbols according to the value of *run* and/or *level* as indicated in Table 7-31. For each category a fixed length code is appended to the variable length code to uniquely identify which value in that category actually occurred. The additional codes are defined in Table B.47 to Table B.49. For decoding group No.1~6, Table B.47 shall be used. For decoding group No.7~12, Table B.48 shall be used. For decoding group No.13~20, Table B.49 shall be used.

3) An “Escape” coded coefficient. In which a value of *level* is fixed length coded as described in 7.16.4.1.3.

The VLC table which is used to decode a variable length code for the current symbol shall be decided from the 12 tables according to the immediately previous symbol which has already been decoded in a zigzag scanning order. After decoding the current symbol, it is used to decide the VLC table for decoding the next symbol from the 12 tables.

The relation of the value of the current symbol and the VLC table used for decoding a variable length code for the next symbol is indicated in Table 7-31.

The relation of the value of the previous symbol and the VLC table for decoding the current symbol is shown in Table 7-31. A variable length code for the current symbol is decoded using one of the 12 tables, and the variable length code indicates the group (0~21) that the value of the current symbol belongs to. Depending on the group, the size of a fixed length code used to identify the value of the current symbol and the VLC table for decoding the next symbol is decided. For the first non DC coefficient in a block the VLC table T[0] shall be used as the initial table.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-31 — Categorizing AC symbol into groups

group no.	symbols of DCT AC coefficients (zigzag scanning order)	the size of an additional code	VLC table for the next symbol
0	EOB	0	---
1	0*1	0	T[1]
2	0*2 to 0*3	1	T[1]
3	0*4 to 0*7	2	T[1]
4	0*8 to 0*15	3	T[1]
5	0*16 to 0*31	4	T[1]
6	0*32 to 0*63	5	T[1]
7	0*1[1,-1]	1	T[2]
8	0*2[1,-1] to 0*3[1,-1]	2	T[2]
9	0*4[1,-1] to 0*7[1,-1]	3	T[2]
10	0*8[1,-1] to 0*15[1,-1]	4	T[2]
11	0*16[1,-1] to 0*31[1,-1]	5	T[2]
12	0*32[1,-1] to 0*63[1,-1]	6	T[2]
13	-1, or 1	1	T[3]
14	-3 to –2, or 2 to 3	2	T[4]
15	-7 to –4, or 4 to 7	3	T[5]
16	-15 to –8, or 8 to 15	4	T[6]
17	-31 to –16, or 16 to 31	5	T[7]
18	-63 to –32, or 32 to 63	6	T[8]
19	-127 to –64, or 64 to 127	7	T[9]
20	-255 to –128, or 128 to 255	8	T[10]
21	$-(2^{\text{bits_per_pixel+dct_precision+3}}-1)$ to –256, or 256 to $(2^{\text{bits_per_pixel+dct_precision+3}}-1)$ (escape code)	bits_per_pixel + dct_precision + 4	T[11]

Note : Table T[0] shall be used for the first non DC coefficient in a block.

The VLC code for group ‘0’ of T[0] for inter macroblocks shall not be assigned because there shall be at least one non zero DCT coefficient in an inter block which is indicated as ‘coded’ by pattern_code[].

The VLC code for group ‘0’~’13’ of T[1] shall not be assigned because there shall be at least one *level* value , the absolute value of which is more than ‘1’, just after group ‘1’~’6’ occurred.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

7.16.4.1.3 Escape coding

Level values, the absolute value of that is more then 255, are encoded by using the escape code. The escape code that is assigned to group No.21 is followed by a fixed length code of the length 'bits_per_pixel+dct_precision+4'. The following equation defines the relation among the fixed length code, the length and *level* to be decoded.

if flc>>(flclen-1) is 1

level = -1 * ((flc^((1<<flclen)-1)) + 1)

else *level* = flc

Where, flc and flclen denote the fixed length code and its length. The symbol ^ denotes XOR in this subclause.

The symbols that can be coded by using a non escape codeword shall not be coded by using the escape codeword. The codeword with the MSB '1' followed by only 0 binary sequence '100...000' is reserved.

7.16.4.2 Inverse scan

This subclause specifies the way in which the one dimensional data, QFS[n] is converted into a two-dimensional array of coefficients denoted by QF[v][u] where u and v both lie in the range of 0 to 7. Let the data at the output of the variable length decoder be denoted by QFS[n] where n is in the range of 0 to 63.

Two scan patterns are defined. The scan that shall be used shall be determined by alternate_scan which is encoded in StudioVideoObjectPlane().

Figure 7-104 defines *scan*[alternate_scan][*v*][*u*] for the case that alternate_scan is zero. Figure 7-105 defines *scan*[alternate_scan][*v*][*u*] for the case that alternate_scan is one.

		u							
		0	1	2	3	4	5	6	7
v	0	0	1	5	6	14	15	27	28
	1	2	4	7	13	16	26	29	42
	2	3	8	12	17	25	30	41	43
	3	9	11	18	24	31	40	44	53
	4	10	19	23	32	39	45	52	54
	5	20	22	33	38	46	51	55	60
	6	21	34	37	47	50	56	59	61
	7	35	36	48	49	57	58	62	63

Figure 7-104 — Definition of *scan*[0][*v*][*u*]

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

		u						
		0	1	2	3	4	5	6	7
v	0	0	4	6	20	22	36	38	52
	1	1	5	7	21	23	37	39	53
	2	2	8	19	24	34	40	50	54
	3	3	9	18	25	35	41	51	55
	4	10	17	26	30	42	46	56	60
	5	11	16	27	31	43	47	57	61
	6	12	15	28	32	44	48	58	62
	7	13	14	29	33	45	49	59	63

Figure 7-105 — Definition of *scan*[1][*v*][*u*]

The inverse scan shall be any process equivalent to the following:

```
for (v=0; v<8; v++)
    for (u=0; u<8; u++)
        QF[v][u] = QFS[scan[alternate_scan][v][u]]
```

NOTE - The scan patterns defined here are often referred to as “zigzag scanning order”.

7.16.4.2.1 Inverse scan for matrix download

When the quantisation matrices are downloaded they are encoded in the bitstream in a scan order that is converted into the two-dimensional matrix used in the inverse quantisation process in an identical manner to that used for coefficients.

For matrix download the scan defined by Figure 7-104 (i.e. *scan*[0][*v*][*u*]) shall always be used.

Let *W*[*w*][*v*][*u*] denote the quantisation matrix in the inverse quantisation process (see 7.16.4.3.2.1), and *W'*[*w*][*n*] denote the matrix as it is encoded in the bitstream. The matrix download shall then be equivalent to the following:

```
for (v=0; v<8; v++)
    for (u=0; u<8; u++)
        W[w][v][u] = W'[w][scan[0][v][u]]
```

7.16.4.3 Inverse quantisation

The two-dimensional array of coefficients, *QF*[*v*][*u*], is inverse quantised to produce the reconstructed DCT coefficients. This process is a multiplication by the base quantiser scale and the quantisation step size. The quantisation step size is modified by two mechanisms; a quantisation matrix is used to modify the step size within a block and a quantiser scale is used in order that the step size can be modified at the cost of only a few bits (as compared to encoding an entire new quantisation matrix). In case of mpeg2_stream==0, the base quantiser scale is used to adjust the dynamic range of the output of the inverse quantisation to the range $[-2^{bits_per_pixel\ +\ 6}, 2^{bits_per_pixel\ +\ 6} - 1]$. In case of mpeg2_stream==1, the base quantiser scale shall not affect the decoding process below.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

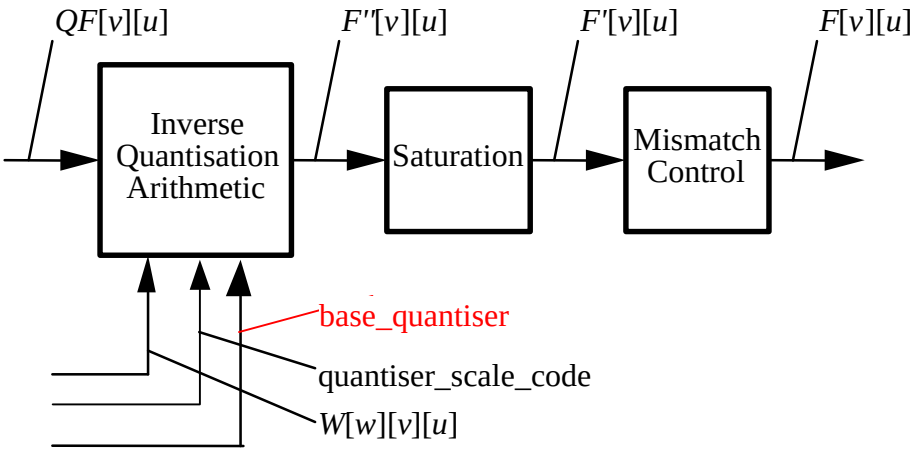


Figure 7-106 — Inverse quantisation process

Figure 7-106 illustrates the overall inverse quantisation process. After the appropriate inverse quantisation arithmetic the resulting coefficients, *F''*[*v*][*u*], are saturated to yield *F'*[*v*][*u*] and then a mismatch control operation is performed to give the final reconstructed DCT coefficients, *F*[*v*][*u*].

NOTE Attention is drawn to the fact that the method of achieving mismatch control in this part of ISO/IEC 14496 is identical to that employed by ISO/IEC 13818-2:2000.

7.16.4.3.1 Intra DC coefficient

The DC coefficients of intra coded blocks shall be inverse quantised in a different manner to all other coefficients.

In intra blocks *F'*[0][0] shall be obtained by multiplying *QF*[0][0] by a constant multiplier, *intra_dc_mult*, (constant in the sense that it is not modified by either the quantisation matrix or the quantiser scale). The multiplier is related to the parameters base_quantiser scale and intra_dc_precision that are encoded in StudioVideoObjectPlane(). Table 7-32 and Table 7-33 specify the relation between intra_dc_precision, base_quantiser scale and *intra_dc_mult*. This relation changes according to the value of mpeg2_stream. The precision of the intra dc coefficient is defined as 'bits_per_pixel+dct_precision+intra_dc_precision'.

Table 7-32 — In case of mpeg2_stream==0:
Reletion among intra_dc_precision, base_quantiser scaleand *intra_dc_mult*

intra_dc_precision	*intra_dc_mult*
0	8 x base_quantiser scale x 8
1	4 x base_quantiser scale x 8
2	2 x base_quantiser scale x 8
3	1 x base_quantiser scale x 8

Table 7-33 — In case of mpeg2_stream==1:
Reletion between intra_dc_precision, *intra_dc_mult*

intra_dc_precision	*intra_dc_mult*
0	8
1	4
2	2
3	1

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The reconstructed DC values are computed as follows.

F''[0][0] = intra_dc_mult* QF[0][0]

7.16.4.3.2 Other coefficients

All coefficients other than the DC coefficient of an intra block shall be inverse quantised as specified in this subclause.

7.16.4.3.2.1 Quantisation matrices

In the 4:2:0 format, two quantisation matrices are used. One shall be used for intra macroblocks and the other for non-intra macroblocks. In the 4:2:2 or 4:4:4 format, four matrices are used allowing different matrices to be used for luminance and chrominance data. Each matrix has a default set of values which may be overwritten by down-loading a user defined matrix as explained in 6.2.13.2.4.

Let the quantisation matrices be denoted by $W[w][v][u]$ where w takes the values 0 to 3 indicating which of the matrices is being used. Table 7-34 summarises the rules governing the selection of w.

Table 7-34 — Selection of *w*

	4:2:0		4:2:2 and 4:4:4	
	luminance (cc = 0)	**chrominance (cc ≠ 0)**	**luminance (cc = 0)**	**chrominance (cc ≠ 0)**
intra blocks (macroblock_intra = 1)	0	0	0	2
non-intra blocks (macroblock_intra = 0)	1	1	1	3

7.16.4.3.2.2 Quantiser scale

The quantiser scale is decoded as a 5 bit fixed length code, quantiser_scale_code. This indicates the appropriate *quantiser_scale* to apply in the inverse quantisation arithmetic.

q_scale_type (encoded in StudioVideoObjectPlane()) indicates which of two mappings between quantiser_scale_code and *quantiser_scale* shall apply. Table 7-35 shows the two mappings between quantiser_scale_code and *quantiser_scale*.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-35 — Relation between *quantiser_scale* and quantiser_scale_code

quantiser_scale_code	quantiser_scale[q_scale_type]	
	q_scale_type = 0	q_scale_type = 1
0	(forbidden)	
1	2	1
2	4	2
3	6	3
4	8	4
5	10	5
6	12	6
7	14	7
8	16	8
9	18	10
10	20	12
11	22	14
12	24	16
13	26	18
14	28	20
15	30	22
16	32	24
17	34	28
18	36	32
19	38	36
20	40	40
21	42	44
22	44	48
23	46	52
24	48	56
25	50	64
26	52	72
27	54	80
28	56	88
29	58	96
30	60	104
31	62	112

7.16.4.3.2.3 Reconstruction formulae

The following equation specifies the arithmetic to reconstruct *F''*[*v*][*u*] from *QF*[*v*][*u*] (for all coefficients except intra DC coefficients) in the case of mpeg2_stream==0.

$$F''[v][u] = ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser_scale \times base_quantiser \times 8) / 32$$

where:

$$k = \begin{cases} 0 & \text{intra blocks / non-intra blocks if dead_zone_disable==1} \\ Sign(QF[v][u]) & \text{non-intra blocks if dead_zone_disable==0} \end{cases}$$

NOTE The above equation uses the "/" operator as defined in 4.1.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This equation changes in the case of mpeg2_stream==1 as following:

$$F''[v][u] = ((2 \times QF[v][u] + k) \times W[w][v][u] \times quantiser_scale) / 32$$

where:

$$k = \begin{cases} 0 & \text{intra blocks} \\ Sign(QF[v][u]) & \text{non-intra blocks} \end{cases}$$

NOTE The above equation uses the "/" operator as defined in 4.1.

7.16.4.3.3 Saturation

The coefficients resulting from the Inverse Quantisation Arithmetic are saturated to lie in the range [$-2^{bits_per_pixel + 6}$, $2^{bits_per_pixel + 6} - 1$] in the case of mpeg2_stream==0. Thus:

$$F'[v][u] = \begin{cases} 2^{bits_per_pixel+6} - 1 & F''[v][u] > 2^{bits_per_pixel+6} - 1 \\ F''[v][u] & -2^{bits_per_pixel+6} \le F''[v][u] \le 2^{bits_per_pixel+6} - 1 \\ -2^{bits_per_pixel+6} & F''[v][u] < -2^{bits_per_pixel+6} \end{cases}$$

The coefficients resulting from the Inverse Quantisation Arithmetic are saturated to lie in the range [$-2^{bits_per_pixel + 3}$, $2^{bits_per_pixel + 3} - 1$] in the case of mpeg2_stream==1. Thus:

$$F'[v][u] = \begin{cases} 2^{bits_per_pixel+3} - 1 & F''[v][u] > 2^{bits_per_pixel+3} - 1 \\ F''[v][u] & -2^{bits_per_pixel+3} \le F''[v][u] \le 2^{bits_per_pixel+3} - 1 \\ -2^{bits_per_pixel+3} & F''[v][u] < -2^{bits_per_pixel+3} \end{cases}$$

7.16.4.3.4 Mismatch control

Mismatch control shall be performed by any process equivalent to the following. Firstly all of the reconstructed, saturated coefficients, *F'[v][u]* in the block shall be summed. This value is then tested to determine whether it is odd or even. If the sum is even then a correction shall be made to just one coefficient; *F*[7][7]. Thus:

$$sum = \sum_{v=0}^{v<8} \sum_{u=0}^{u<8} F'[v][u]$$

$$F[v][u] = F'[v][u] \text{ for all } u, v \text{ except } u = v = 7$$

$$F[7][7] = \begin{cases} F'[7][7] & \text{if } sum \text{ is odd} \\ \left\{ \begin{array}{ll} F'[7][7] - 1 & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1 & \text{if } F'[7][7] \text{ is even} \end{array} \right\} & \text{if } sum \text{ is even} \end{cases}$$

NOTE 1 It may be useful to note that the above correction for *F*[7][7] may simply be implemented by toggling the least significant bit of the twos complement representation of the coefficient. Also since only the "oddness" or "evenness" of the *sum* is of interest an exclusive OR (of just the least significant bit) may be used to calculate "*sum*".

NOTE 2 Warning. Small non-zero inputs to the IDCT may result in zero output for compliant IDCTs. If this occurs in an encoder, mismatch may occur in some pictures in a decoder that uses a different compliant IDCT. An encoder should avoid this problem and may do so by checking the output of its own IDCT. It should ensure that it never inserts any non-zero coefficients into the bitstream when the block in question reconstructs to zero through its own IDCT function. If this action is not taken by the encoder, situations can arise where large and very visible mismatches between the state of the encoder and decoder occur.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.4.3.5 Summary of inverse quantisation process

In summary, the inverse quantisation process is any process numerically equivalent to:

```
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if ( (u==0) && (v==0) && (macroblock_intra) ) {
            F''[v][u] = intra_dc_mult * QF[v][u];
        } else {
            if ( macroblock_intra ) {
                F''[v][u] = ( QF[v][u] * W[0][v][u] * quantiser_scale * base_quantiser scale * 8 * 2 ) / 32;
            } else {
                F''[v][u] = ( ( ( QF[v][u] *  2 ) + Sign(QF[v][u]) ) * W[1][v][u]
                                                        * quantiser_scale * base_quantiser scale * 8 ) / 32;
            }
        }
    }
}
sum = 0;
for (v=0; v<8;v++) {
    for (u=0; u<8;u++) {
        if ( F''[v][u] > 2^(bits_per_pixel + 6) – 1 ) {
            F'[v][u] = 2^(bits_per_pixel + 6) – 1;
        } else {
            if ( F''[v][u] < -2^(bits_per_pixel + 6) ) {
                F'[v][u] = -2^(bits_per_pixel + 6);
            } else {
                F'[v][u] = F''[v][u];
            }
        }
    sum = sum + F'[v][u];
    F[v][u] = F'[v][u];
    }
}
if ((sum & 1) == 0) {
    if ((F[7][7] & 1) != 0) {
        F[7][7] = F'[7][7] - 1;
    } else {
        F[7][7] = F'[7][7] + 1;
    }
}
```

7.16.4.4 Inverse DCT

Once the DCT coefficients, F[u][v] are reconstructed, an IDCT transform that conforms to the specifications of Annex A shall be applied to obtain the inverse transformed values, $f[y][x]$. In the case of mpeg2_stream==0, the decimal point of F[u][v] is shifted 3bits to the left in the binary scale in order to adjust the decimal point of the IDCT input. In the case of mpeg2_stream==1, the reconstructed coefficients are directly input to an IDCT function without the shift process. The inverse transformed values shall be saturated so that: $-2^{bits_per_pixel} \leq f[y][x] \leq 2^{bits_per_pixel} - 1$, for all x, y.

7.16.4.4.1 Non-coded blocks and skipped macroblocks

In a macroblock that is not skipped, if pattern_code[i] is one for a given block in the macroblock, then coefficient data is included in the bitstream for that block. This is decoded as specified in the preceding clauses.

However, if pattern_code[i] is zero, or if the macroblock is skipped (not_coded==1), then that block contains no coefficient data. The sample domain coefficients f[y][x] for such a block shall all take the value zero.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.5 Texture decoding from DPCM redisuals

If the DPCM mode is selected each non-transparent block is decoded as follows. First the **rice_parameter** field is decoded. It indicates the length of each subsequent **rice_suffix_code** field. The block is scanned line by line, row by row, and the following process is applied to each pixel. An unsigned residual is retrieved by a combination of the **rice_prefix_code**, **rice_suffix_code** and **dpcm_residual** fields. If **rice_prefix_code** indicates the escape mode, the unsigned residual is given by **dpcm_residual**. Otherwise the unsigned residual is given by (**rice_prefix_code**<<**rice_parameter**)+**rice_suffix_code**. The residuals that can be coded by using a combination of rice_prefix_code and rice_suffix_code shall not be coded by using dpcm_residual.

Then the unsigned residual is mapped to a signed residual according to the following transformation:

$x \rightarrow x>>1$ if x is even

$x \rightarrow -x>>1$ if x is odd

A prediction is computed based on the pixels that lie directly to the left 'a', directly above 'b', and directly left above 'c'. If any of these pixels lies outside of the block boundary it is given the default value of $2^{\mathbf{bits_per_pixel}-1}$. The prediction 'p' is computed as:

p = a+b-c

if (p < min(a,b)) p = min(a,b)

if (p > max(a,b)) p = max(a,b)

A second prediction p2 is computed as:

p2 = (min(a,b,c)+max(a,b,c)) / 2

If p2 is equal to p, p2 is assigned the value of **block_mean**.

If p2 is larger than p, the sign of the residual is inverted. Otherwise no operation is applied to the residual.

Finally the reconstructed pixel value, d''[y][x], is obtained by adding the signed residual and the prediction p, modulo $2^{\mathbf{bits_per_pixel}}$.

7.16.6 Shape decoding

Binary shape decoding uses a block-based representation. The primary data structure used is denoted as the binary alpha block (bab). The bab is a square block of binary valued pixels representing the opacity/transparency for the pixels in a specified block-shaped spatial region of size 16x16 pels. In fact, each bab is co-located with each texture macroblock.

7.16.6.1 Higher syntactic structures

7.16.6.1.1 VOL decoding

If video_object_layer_shape is equal to '00' then no binary shape decoding is required. Otherwise, binary shape decoding is carried out.

7.16.6.1.2 VOP decoding

If video_object_layer_shape is not equal to '00' then, for each subsequent VOP, the dimensions of the bounding rectangle of the reconstructed VOP are obtained from:

- vop_width
- vop_height

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If these decoded dimensions are not multiples of 16, then the values of vop_width and vop_height are rounded up to the nearest integer, which is a multiple of 16. If vop_structure is decoded as field structure, both values are definded in the absolute field coordinates.

Additionally, in order to facilitate motion compensation, the horizontal and spatial position of the VOP are obtained from:

- vop_horizontal_mc_spatial_ref
- vop_vertical_mc_spatial_ref

These spatial references may be different for each VOP. The absolute frame coordinate system must be used for all frame VOPs, while the absolute field coordinate system must be used for all field VOPs. Additionally, the decoded spatial references must have an even value.

Once the above elements have been decoded, the binary shape decoder may be applied to decode the shape of each macroblock within the bounding rectangle.

7.16.6.2 Macroblock decoding

The shape information for each macroblock residing within the bounding rectangle of the VOP is decoded into the form of a 16x16 bab.

7.16.6.2.1 Mode decoding

Each bab belongs to one of five types listed in Table 7-36. The type information is given by the bab_type field which influences decoding of further shape information.

Table 7-36 — List of bab types

bab_type	Semantic	Used in
0	MVs==0 && No Update	P-VOPs
1	MVs!=0 && No Update	P-VOPs
2	Transparent	All VOP types
3	Opaque	All VOP types
4	HHC	All VOP types

7.16.6.2.1.1 I-VOPs

In this specification (Studio Profile), only five bab_types are adopted independently of vop_type.

Suppose that f(x,y) is the bab_type of the bab located at (x,y), where x is the BAB column number and y is the BAB row number. The code word for the bab_type at the position (i,j) is determined as follows. A context C is computed from a previously decoded bab_type.

C = f(i-1,j)-2

If `f(x,y)` references a bab outside the current VOP, bab_type is assumed to be transparent for that bab (i.e. `f(x,y)=2`). bab_type of the bab outside the current slice is also assumed to be transparent. The VLC used to decode bab_type for the current bab is switched according to the value of the context C. This context-switched VLC table is given in Table B.75. The context C shall be set to zero for all macroblocks in a VOP in case of intra_predictors_reset==1 or for a macroblock located at the start of a slice.

If the type of the bab is transparent, then the current bab is filled with zero (transparent) values. A similar procedure is carried out if the type is opaque, where the reconstructed bab is filled with values of 255 (opaque). For both transparent and opaque types, no further decoding of shape-related data is required for the current bab. Otherwise further decoding steps are necessary, which is called HHC. Decoding for HHC is described in subclause 7.16.6.2.5.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.6.2.1.2 P-VOPs

The decoding of the current bab_type is dependent on the bab_type of the co-located bab in the reference VOP. The reference VOP is a forward reference VOP. The forward reference VOP is defined as the most recent non-empty (i.e. vop_coded != 0) I- or P-VOP in the past. If the current VOP is a P-VOP, the forward reference VOP is selected as the reference VOP.

If the current VOP is a field VOP of the interlaced sequence, the forward reference VOP is defined as the most recent non empty I- or P- field VOP of the same position (top or bottom field) in the past. When the second field of the first frame in GOV is coded as P-VOP, the reference VOP is the first field coded as I-VOP.

If the sizes of the current and reference VOPs are different, some babs in the current VOP may not have a co-located equivalent in the reference VOP. Therefore the bab_type matrix of the reference VOP is manipulated to match the size of the current VOP. Two rules are defined for that purpose, namely a cut rule and a copy rule:

- ***cut rule***. If the number of lines (respectively columns) is smaller in the current VOP than in the reference VOP, the bottom lines (respectively rightmost columns) are eliminated from the reference VOP such that both VOP sizes match.

- ***copy rule***. If the number of lines (respectively columns) is larger in the current VOP than in the reference VOP, the bottom line (respectively rightmost column) is replicated as many times as needed in the reference VOP such that both VOP sizes match.

An example is shown in Figure 7-107 where both rules are applied.

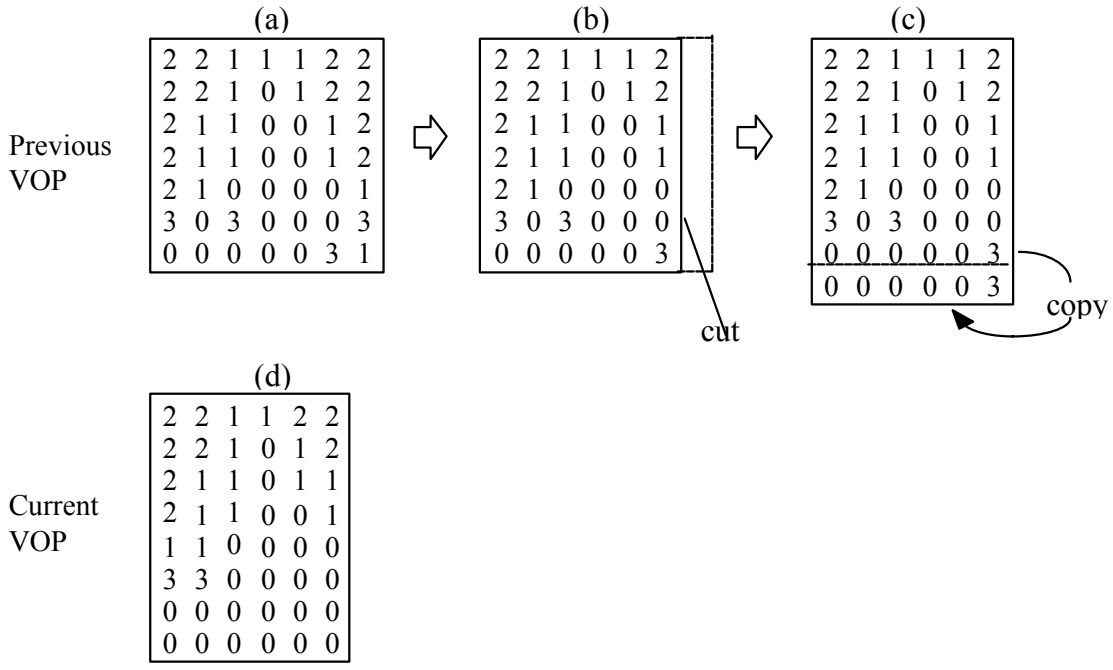


Figure 7-107 — Example of size fitting between current VOP and reference VOP. The numbers represent the type of each bab

The VLC to decode the current bab_type is switched according to the bab_type value of the co-located bab in the reference VOP. These context-switched VLC tables for P-VOPs are given in Table B.76. If the type of the bab is transparent, then the current bab is filled with zero (transparent) values. A similar procedure is carried out if the type is opaque, where the reconstructed bab is filled with values of 255 (opaque). For both transparent and opaque types, no further decoding of shape-related data is required for the current bab. Otherwise further decoding steps are necessary, as listed in Table 7-37. Decoding for motion compensation is described in subclause 7.16.6.2.4, and HHC decoding in subclause 7.16.6.2.5.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-37 — Decoder components applied for each type of bab

bab_type	Motion compensation	HHC
0	yes	no
1	yes	no
2	no	no
3	no	no
4	no	yes

7.16.6.2.2 Binary alpha block motion compensation

Motion Vector of a shape (MVs) is used for motion compensation (MC) of the shape. The value of MVs is reconstructed as described in subclause 7.16.6.2.3. Integer pixel motion compensation is carried out on a 16x16 block basis according to subclause 7.16.6.2.4.

If bab_type is MVs==0 && No Update or MVs!=0 && No Update then the motion compensated bab is taken to be the decoded bab, and no further decoding of the bab is necessary. Otherwise, HHC decoding is required.

7.16.6.2.3 Motion vector decoding

If bab_type indicates that MVs!=0, then mvs_x and mvs_y are VLC decoded. For decoding mvs_x, the VLC given in Table B.77 is used. The same table is used for decoding mvs_y, unless the decoded value of mvs_x is zero. If mvs_x == 0, the VLC given in Table B.78 is used for decoding mvs_y. If bab_type indicates that MVs==0, then both mvs_x and mvs_y are set to zero.

7.16.6.2.4 Motion compensation

For inter mode babs (bab_type = 0 or 1), motion compensation is carried out by simple MV displacement according to the MVs.

Specifically, when bab_type is equal to 0 or 1 i.e. for the no-update modes, a displaced block of 16x16 pixels is copied from the binary alpha map of the previously decoded I or P VOP for which vop_coded is not equal to ‘0’. If the displaced position is outside the bounding rectangle, then these pixels are assumed to be “transparent”.

If the macroblock is in the field VOP, a displayed block is copied from previously decoded field I-VOP of the same position (top or bottom field). When the second field of the first frame in GOV is coded as P-VOP, a displayed block is copied from the first field coded as I-VOP.

7.16.6.2.5 HHC decoding

If an inferior_symbol_macroblock is equal to ‘0’ then the inferior symbol means transparent and the superior symbol means opaque. Otherwise, the inferior symbol means opaque and the superior symbol means transparent. After the inferior symbol is decided, the pattern of inferior symbol in the block is decoded as the following hierarchical process.

7.16.6.2.5.1 Coded binary block pattern

The index of cbbp is obtained by decoding the variable length code to produce coded_binary_block_pattern[i] (i=0,1,2,3). If coded_binary_block_pattern[i] is one for a given block in the macroblock then the data of the block layer is included in the bitstream for that block. However, if coded_binary_block_pattern[i] is zero, then that block contains no inferior symbol for the macroblock. The binary alpha coefficients for such shape blocks shall all take the value equal to the superior symbol for the macroblock.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.6.2.5.2 Binary block decoding

For the block whose coded_binary_block_pattern[I] is one, the data of the block layer is decoded. The inferior and the superior symbol for the block is indicated by inferior_symbol_block. If scan_direction is equal to 0, the block is divided into horizontal rows. If scan_direction is equal to 1, the block is divided into vertical columns. If backward_read_flag is set to '0', lbp corresponds to the pixel pattern from left to right pixel in scan_direction='0' or from upper to lower pixel in scan_direction='1'. If backward_read_flag is set to '1', lbp corresponds to the pixel pattern from right to left pixel in scan_type='0' or from lower to upper pixel in scan_type='1'.

The index of clp is obtained by decoding the variable length code to produce coded_line_pattern[i] (I=0,1,2, ,7). If coded_line_pattern[i] is one for a given line in the block then lbp information is included in the bitstream for that line. However, if coded_line_pattern[i] is zero, then that line contains no inferior symbol for the block. The binary alpha coefficient for such shape lines shall all take the value equal to the superior symbol for the block.

The index of lbp is obtained by decoding the variable length code to produce line_bit_pattern[i] (I=0,1,2, ,7). If backward_read_flag is set to '1', line_bit_pattern[i] is set to line_bit_pattern[7-i]. Note that line_bit_pattern[i] equal to '1' means the pixel is the inferior symbol for the block. Otherwise, line_bit_pattern[i] is equal to '1' means the pixel is the superior symbol for the block.

After all pixels are decoded, the inferior symbol and superior symbol are set to either opaque or transparent symbol, respectively.

7.16.6.3 Grayscale Shape Decoding

Grayscale alpha plane decoding is achieved by the separate decoding of a support region and the values of the alpha channel. The support region is transmitted by using the binary shape as described above. The alpha values are transmitted as texture data with arbitrary shape, using almost the same coding method as is used for the luminance texture channel.

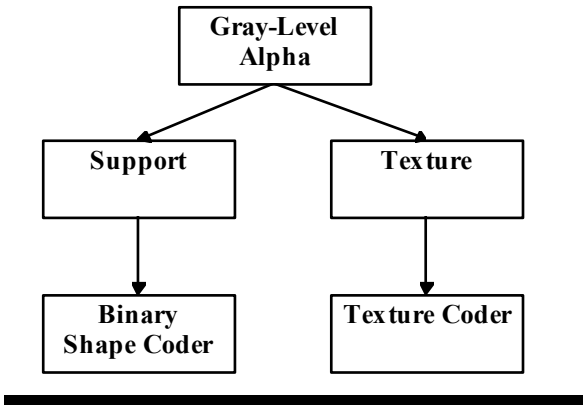


Figure 7-108 — Grayscale shape coding

All samples which are indicated to be transparent by the binary shape data must be set to the value of minimum_alpha_level[i](transparent) in the decoded grayscale alpha plane i. Within the VOP, alpha samples have the values produced by the grayscale alpha decoding process. Decoding of binary shape information is not dependent on the decoding of grayscale alpha. The alpha values are decoded into 16x16 macroblocks in the same way as the luminance channel (see subclause 7.16.4, 7.16.5 and 7.16.7). The 16x16 blocks of alpha values are referred to as alpha macroblocks hereafter. The data for each alpha macroblock is present in the bitstream immediately following the texture data for the corresponding texture macroblock. Any aspect of alpha decoding that is not covered in this document should be assumed to be the same as for the decoding of luminance.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.6.3.1 Grayscale Alpha COD Modes

When decoding grayscale alpha macroblocks, CODA is first encountered and indicates the coding status for alpha. It is important to understand that the macroblock syntax elements for alpha are still present in the bitstream for inter macroblocks even if the texture syntax elements indicate “not-coded” (not_coded=’1’). In this respect, the decoding of the alpha and texture data are independent.

For macroblocks which are completely transparent (indicated by the binary shape coding), no alpha syntax elements are present and the grayscale alpha samples must all be set to the value of minimum_alpha_level[i] (transparent). If CODA=”all opaque” (intra, inter macroclocks) or CODA = "residual all zero" (inter macroblocks) then no more alpha data is present. Otherwise, other alpha syntax elements follow, i.e. alpha texture data encoded as either DCT coefficients which are coded and non-transparent or DPCM residuals, as is the case for regular luminance macroblock texture data.

When CODA=”all opaque”, the corresponding decoded alpha macroblock is filled with the value of maximum_alpha_level[i]. This value will be called AlphaOpaqueValue[i].

7.16.6.3.2 Intra Macroblocks coded as DPCM residuals

In case of alpha_compression_mode[i]==1(DPCM mode), the grayscale alpha data shall be decoded as DPCM residuals. DPCM mode can be selected independently of intra/inter mode of the texture data.

7.16.6.3.3 Intra Macroblocks coded as DCT coefficients

When the texture is encoded as DPCM residuals (compression_mode==1) or intra DCT coefficients (compression_mode==0 && macroblock_intra==1), and if alpha_compression_mode[i]==0, the grayscale alpha data is encoded as intra DCT residual.

The intra DC value is decoded in the same way as for luminance.

The DC predictor is used in the same way as for luminance. However, when coda_i indicates that a macroblock is all opaque, the predictor shall be reset so that the next intra block to be decoded is correctly decoded. The reset value is defined as;

AlphaOpaqueValue[i] * (2^(alpah_dct_precision[i]+alpha_intra_dc_precision[i]))

AlphaOpaqueValue[i] is described in subclause 7.16.6.3.1.

7.16.6.3.4 Inter Macroblocks and Motion Compensation

When the texture data in a P-VOP is encoded as inter macroblock (compression_mode==0 && macroblock_intra==0), and if alpha_compression_mode[i]==0, the alpha macroblock shall be decoded as inter macroblock.

Motion compensation is carried out for inter macroblocks, using the 16x16 or 16x8 luminance motion vectors, in the same way as for luminance data. Where the luminance motion vectors are not present because the texture macroblock is skipped, the exact same style of non-coded motion compensation used for luminance is applied to the alpha data. Note that this does not imply that the alpha macroblock is skipped, because an error signal to update the resulting motion compensated alpha macroblock may still be present if indicated by coda_pb.

cbpa is defined in Table B.83 and alpha_pattern_code is derived from cbps.

Alpha inter DCT coefficients are decoded in the same way as the luminance coefficients. The only exception is that the inverse quantisation process in 7.16.4.2.3 shall perfom as if dead_zone_disable is set to zero independently of the value of dead_zone_disable.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.6.3.5 Method to be used when blending with greyscale alpha signal

The following explains the blending method to be applied to the video object in the compositor, which is controlled by the composition_method flag and the linear_composition flag. The linear_composition flag is informative only, and the decoder may ignore it and proceed as if it had the value 0. However, it is normative that the composition_method flag be acted upon.

The descriptions below show the processing taking place in YUV space; note that the processing can of course be implemented in RGB space to obtain equivalent results.

saturation

The alpha signals are saturated to lie in the range [Tr, Op]. Thus:

```
        { Op       if alpha > Op
alpha = { alpha          if  Tr <= alpha <= Op
        { Tr       if alpha < Tr
```

composition_method=0 (cross-fading)

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

```
Pyuv   =      ( (R - (Nalpha-Tr)) * Myuv + ((Nalpha-Tr) * Nyuv ) ) / R
Palpha =      Op
```

composition_method=1 (Additive mixing)

If layer N, with an n-bit alpha signal, is overlaid over layer M to generate a new layer P, the composited Y, U, V and alpha values are:

```
      { Myuv                                  ..... Nalpha = Tr
Pyuv  = {
      { (Myuv - BLACK)  -  ( (Myuv - BLACK) * (Nalpha-Tr) ) / R+ Nyuv     ..... Nalpha > Tr
```

(this is equivalent to Pyuv = Myuv*(1-alpha) + Nyuv, taking account of black level and the fact that the video decoder does not produce an output in areas where alpha=0)

```
Palpha =      Nalpha + Malpha -Tr - ((Nalpha-Tr)*(Malpha-Tr)) / R
```

where

BLACK is the common black value of foreground and background objects.

Tr: level for complete transparency

Op: level for complete opacity

R: = Op - Tr (dynamic range of alpha signal)

AnnexNC NOTE The compositor must convert foreground and background objects to the same black value and signal range before composition. The black level of each video object is specified by the video_range bit in the video_signal_type field, or by the default value if the field is not present. (The RGB values of synthetic objects are specified in a range from 0 to 1, as described in ISO/IEC 14496-1).

linear_composition = 0: The compositing process is carried out using the video signal in the format from which it is produced by the video decoder, that is, without converting to linear signals. Note that because video signals are usually non-linear (“gamma-corrected”), the composition will be approximate.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

linear_composition = 1: The compositing process is carried out using linear signals, so the output of the video decoder is converted to linear if it was originally in a non-linear form, as specified by the video_signal_type field. Note that the alpha signal is always linear, and therefore requires no conversion.

7.16.6.4 Multiple Auxiliary Component Decoding

Auxiliary components are defined for the VOP on a pixel-by-pixel basis, and contain data related to the video object, like disparity, depth, additional texture. Up to 3 auxiliary components (including the grayscale shape) are possible. The number and type of these components is indicated by the video_object_layer_shape_extension given in Table 6-84. For example, a value '0000' indicates the grayscale (alpha) shape. The same support region as described in 7.16.6.2 is used for all auxiliary components, and the decoding procedure is the same as described in 7.16.6.3.

7.16.7 Motion compensation decoding

The motion compensation process forms predictions from previously decoded VOPs which are combined with the coefficient data (from the output of the IDCT) in order to recover the final decoded samples.

Up to two separate predictions are formed for each block which are combined together to form the final prediction block p[y][x].

In the case of intra coded macroblocks no prediction is formed so that p[y][x] will be zero. The saturation shown in Figure 7-102 is still required in order to remove prohibited values from f[y][x].

In the case where a block is not coded, either because the entire macroblock is skipped or the specific block is not coded there is no coefficient data. In this case f[y][x] is zero and the decoded samples are the prediction, p[y][x].

All the processes except for resetting motion vector predictors in 7.16.7.4.4 are never executed when a macroblock is completely transparent.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

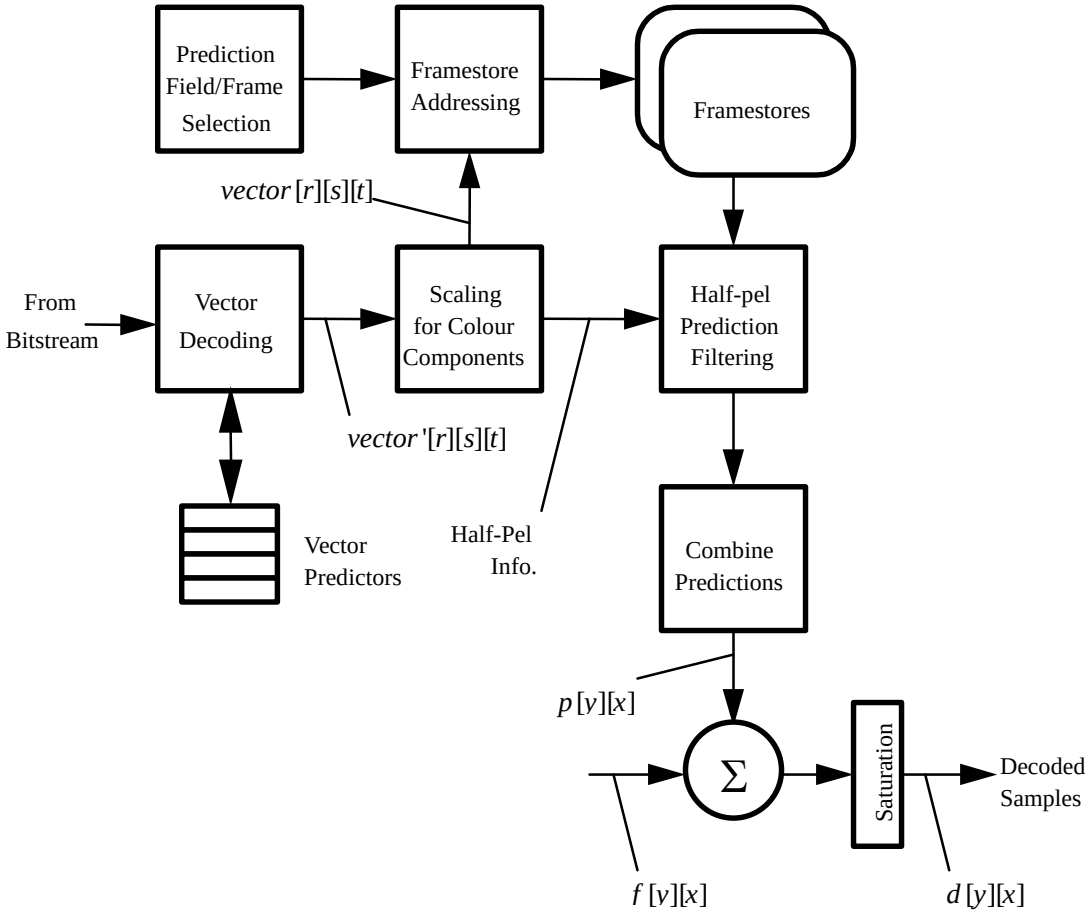


Figure 7-109 — Simplified motion compensation process

7.16.7.1 Motion compensation decoding of arbitrary shaped VOP

In order to perform motion compensated prediction on a per VOP basis, a special padding technique, i.e. the macroblock-based repetitive padding, is applied for the reference VOP. The details of these techniques are described in the following subclauses.

Since a VOP may have arbitrary shape, and this shape can change from one instance to another, conventions are necessary to ensure the consistency of the motion compensation process.

The absolute (frame or field) coordinate system is used for referencing every VOP. At every given instance, a bounding rectangle that includes the shape of that VOP, as described in subclause 7.16.4, is defined. The left and top corner, in the absolute coordinates, of the bounding rectangle is decoded from VOP spatial reference. Thus, the motion vector for a particular feature inside a VOP, e.g. a macroblock, refers to the displacement of the feature in absolute coordinates. No alignment of VOP bounding rectangles at different time instances is performed.

7.16.7.1.1 Padding process

The padding process defines the values of luminance and chrominance samples outside the VOP for prediction of arbitrarily shaped objects. Figure 7-110 shows a simplified diagram of this process.

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

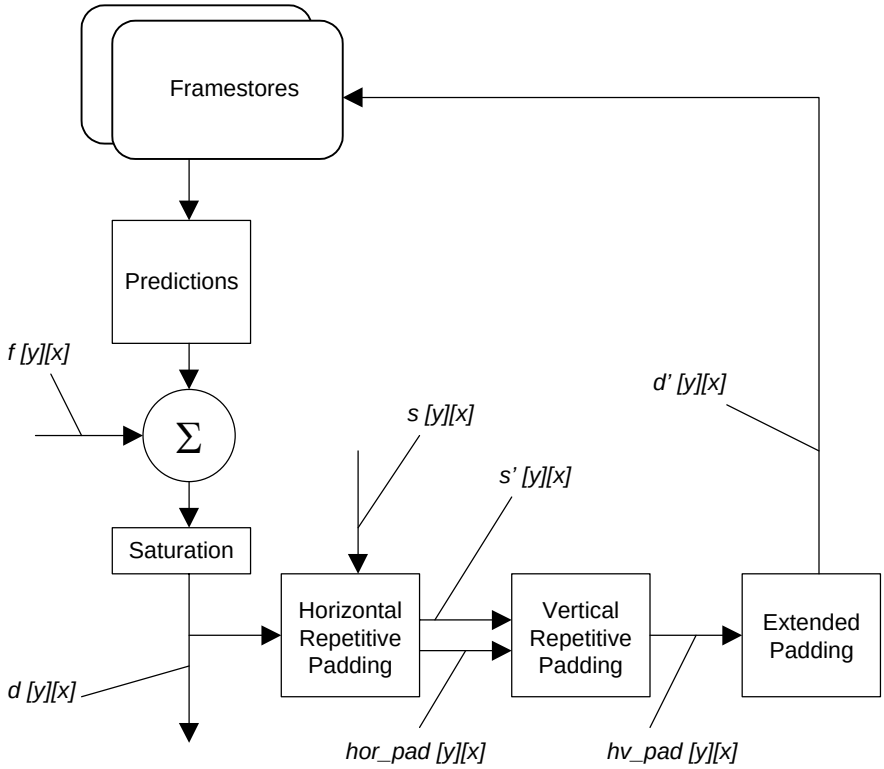


Figure 7-110 — Simplified padding process

A decoded macroblock *d[y][x]* is padded by referring to the corresponding decoded shape block *s[y][x]*. The luminance component is padded per 16 x 16 samples, while the chrominance components are padded per 8 x 8 samples for 4:2:0 format, 16 x 8 samples for 4:2:2 format, 16 x 16 samples for 4:4:4 format. A macroblock that lies on the VOP boundary (hereafter referred to as a boundary macroblock) is padded by replicating the boundary samples of the VOP towards the exterior. This process is divided into horizontal repetitive padding and vertical repetitive padding. The remaining macroblocks that are completely outside the VOP (hereafter referred to as exterior macroblocks) are filled by extended padding.

NOTE The padding process is applied to all macroblocks inside the bounding rectangle of a VOP. The bounding rectangle of the luminance component is defined by vop_width and vop_height extended to a multiple of 16, while that of the chrominance components is defined by (vop_width>>1) and (vop_height>>1) extended to multiple of 8.

7.16.7.1.1.1 Horizontal repetitive padding

Each sample at the boundary of a VOP is replicated horizontally to the left and/or right direction in order to fill the transparent region outside the VOP of a boundary macroblock. If there are two boundary sample values for filling a sample outside of a VOP, the two boundary samples are averaged (//2).

hor_pad[y][x] is generated by any process equivalent to the following example. For every line with at least one shape sample *s[y][x]* == 1(inside the VOP) :

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
for (x=0; x<N; x++) {
    if (s[y][x] == 1) { hor_pad[y][x] = d[y][x]; s'[y][x] = 1; }
    else {
        if ( s[y][x'] == 1 && s[y][x"] == 1)  {
            hor_pad[y][x] = (d[y][x']+ d[y][x"])//2;
            s'[y][x] = 1;
        } else if ( s[y][x'] == 1 ) {
            hor_pad[y][x] = d[y][x']; s'[y][x] = 1;
        } else if ( s[y][x"] == 1 ) {
            hor_pad[y][x] = d[y][x"]; s'[y][x] = 1;
        }
    }
}
```

where *x'* is the location of the nearest valid sample (*s[y][x']* == 1) at the VOP boundary to the left of the current location *x*, *x"* is the location of the nearest boundary sample to the right, and *N* is the number of samples of a line in a macroblock. *s'[y][x]* is initialized to 0.

7.16.7.1.1.2 Vertical repetitive padding

The remaining unfilled transparent horizontal samples (where *s'[y][x]* == 0) from subclause 7.16.7.1.1.1 are padded by a similar process as the horizontal repetitive padding but in the *vertical* direction. The samples already filled in subclause 7.16.7.1.1.1 are treated as if they were inside the VOP for the purpose of this vertical pass.

hv_pad[y][x] is generated by any process equivalent to the following example. For every column of *hor_pad[y][x]* :

```
for (y=0; y<M; y++) {
    if (s'[y][x] == 1)
        hv_pad[y][x] =hor_pad[y][x];
    else {
        if ( s'[y'][x] == 1 && s'[y"][x] == 1 )
            hv_pad[y][x] = (hor_pad[y'][x] + hor_pad[y"][x])//2;
        else if ( s'[y'][x] == 1 )
            hv_pad[y][x] = hor_pad[y'][x];
        else if (s'[y"][x] == 1 )
            hv_pad[y][x] = hor_pad[y"][x];
    }
}
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

where *y'* is the location of the nearest valid sample (*s'[y'][x]* == 1) above the current location *y* at the boundary of *hv_pad*, *y"* is the location of the nearest boundary sample below *y*, and *M* is the number of samples of a column in a macroblock.

7.16.7.1.1.3 Extended padding

Exterior macroblocks immediately next to boundary macroblocks are filled by replicating the samples at the border of the boundary macroblocks. Note that the boundary macroblocks have been completely padded in subclause 7.16.7.1.1.1 and subclause 7.16.7.1.1.2. If an exterior macroblock is next to more than one boundary macroblock, one of the macroblocks is chosen, according to the following convention, for reference.

The boundary macroblocks surrounding an exterior macroblock are numbered in priority according to Figure 7-111. The exterior macroblock is then padded by replicating upwards, downwards, leftwards, or rightwards the row of samples from the horizontal or vertical border of the boundary macroblock having the largest priority number.

The remaining exterior macroblocks (not located next to any boundary macroblocks) are filled with $2^{bits_per_pixel-1}$. For 8-bit luminance component and associated chrominance this implies filling with 128.

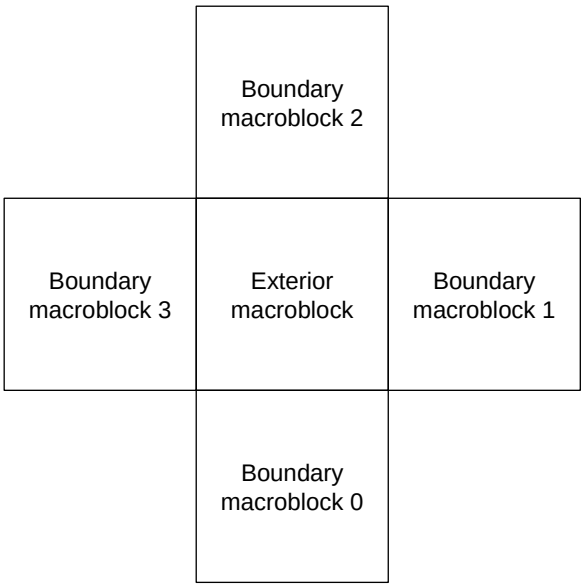


Figure 7-111 — Priority of boundary macroblocks surrounding an exterior macroblock

7.16.7.1.1.4 Padding for chrominance components

Chrominance components are padded according to subclauses 7.16.7.1.1.1 through 7.16.7.1.1.3. The padding is performed by referring to a shape block generated by decimating the shape block of the corresponding luminance component. This decimating of the shape block is performed by the subsampling process described in subclause 6.1.3.6.

7.16.7.1.1.5 Padding of interlaced macroblocks

Macroblocks of an interlaced VOP (progressive_sequence = 0) are padded according to 7.16.7.1.1.1 through 7.16.7.1.1.3. The vertical padding of the luminance component, however, is performed for each field independently. A sample outside of a VOP is therefore filled with the value of the nearest boundary sample of the same field. Completely transparent blocks are padded with $2^{bits_per_pixel-1}$. Chrominance components of interlaced VOP are padded according to subclause 7.16.7.1.1.4, however, based on fields. The padding method described in this subclause is not used outside the bounding rectangle of the VOP.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.7.2 Prediction modes

There are two major classifications of the prediction mode:

— field prediction; and

— frame prediction.

In field prediction, predictions are made independently for each field by using data from one or more previously decoded fields. Frame prediction forms a prediction for the frame from one or more previously decoded frames. It must be understood that the fields and frames from which predictions are made may themselves have been decoded as either field VOPs or frame VOPs.

Within a field VOP all predictions are field predictions. However in a frame VOP either field predictions or frame predictions may be used (selected on a macroblock-by macroblock basis).

In addition to the major classification of field or frame prediction a special prediction mode is used:

- *16x8 motion compensation* - In which two motion vectors are used for each macroblock. The first motion vector is used for the upper 16x8 region, the second for the lower 16x8 region. In this specification 16x8 motion compensation shall only be used with field VOPs.

7.16.7.3 Prediction field and frame selection

The selection of which fields and frames shall be used to form predictions shall be made as detailed in this clause.

7.16.7.3.2 Field prediction

In P-VOPs, the two reference fields from which predictions shall made are the most recently decoded reference top field and the most recently decoded reference bottom field. The simplest case illustrated in Figure 7-112 shall be used when predicting the first VOP of a coded frame or when using field prediction within a frame-VOP. In these cases the two reference fields are part of the same reconstructed frame.

NOTES -

1 The reference fields may themselves have been reconstructed from two field-VOPs or a single frame-VOP.

2 In the case of predicting a field VOP, the field being predicted may be either the top field or the bottom field.

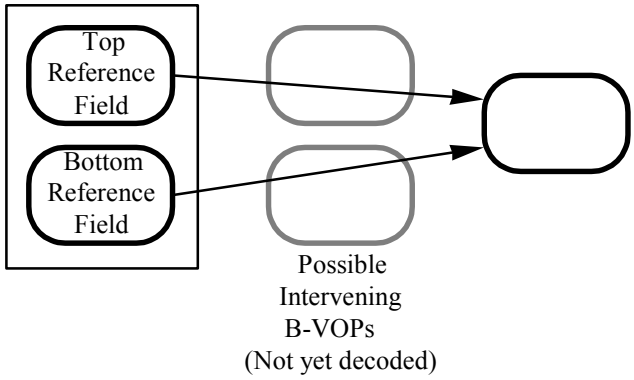


Figure 7-112 — Prediction of the first field or field prediction in a frame-VOP

The case when predicting the second field VOP of a coded frame is more complicated because the two most recently decoded reference fields shall be used, and in this case, the most recent reference field was obtained from decoding the first field VOP of the coded frame. Figure 7-113 illustrates the situation when this second VOP is the bottom field. Figure 7-114 illustrates the situation when this second VOP is the top field.

NOTE The earlier reference field may itself have been reconstructed by decoding a field VOP or a frame VOP.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

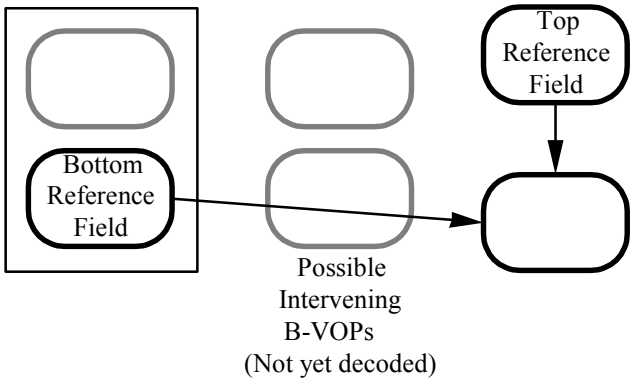


Figure 7-113 — Prediction of the second field-VOP when it is the bottom field

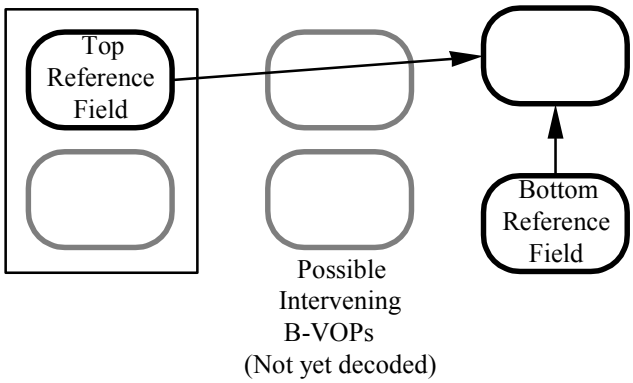


Figure 7-114 — Prediction of the second field-VOP when it is the top field

7.16.7.3.3 Frame prediction

In P-VOPs prediction shall be made from the most recently reconstructed reference frame. This is illustrated in Figure 7-115.

NOTE The reference frame may itself have been reconstructed from two field VOPs or a single frame VOP.

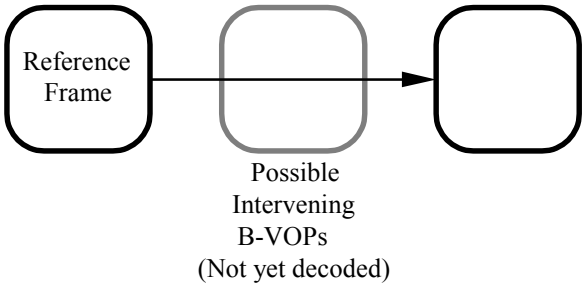


Figure 7-115 — Frame-prediction for P-VOPs

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.7.4 Motion vectors

Motion vectors are coded differentially with respect to previously decoded motion vectors in order to reduce the number of bits required to represent them. In order to decode the motion vectors the decoder shall maintain two motion vector predictors (each with a horizontal and vertical component) denoted *PMV*[*r*][*s*][*t*]. For each prediction, a motion vector, *vector'*[*r*][*s*][*t*] is first derived. This is then scaled depending on the sampling structure (4:2:0, 4:2:2 or 4:4:4) to give a motion vector, *vector*[*r*][*s*][*t*], for each colour component. The meanings associated with the dimensions in this array are defined in Table 7-38.

Table 7-38 — Meaning of indices in *PMV*[*r*][*s*][*t*], *vector*[*r*][*s*][*t*] and *vector'*[*r*][*s*][*t*]

	0	1
r	First motion vector in Macroblock	Second motion vector in Macroblock
s	Forward motion Vector	unused
t	Horizontal Component	Vertical Component

7.16.7.4.1 Decoding the motion vectors

Each motion vector component, *vector'*[*r*][*s*][*t*], shall be calculated by any process that is equivalent to the following one. Note that the motion vector predictors shall also be updated by this process.

```
r_size = vop_fcode[s][t] - 1
f = 1 << r_size
high = ( 16 * f ) -  1;
low = ( (-16) * f );
range = ( 32 * f );

if ( (f  == 1) || (motion_code[r][s][t] == 0) )
        delta = motion_code[r][s][t] ;
else {
        delta = ( ( Abs(motion_code[r][s][t]) - 1 ) * f ) + motion_residual[r][s][t] + 1;
        if (motion_code[r][s][t] < 0)
            delta = - delta;
}

prediction = PMV[r][s][t];
if ( (mv_format == "field") && (t==1) && (VOP_structure == "Frame VOP") )
        prediction = PMV[r][s][t] //// 2;

vector'[r][s][t]= prediction + delta;
if (vector'[r][s][t] < low )
        vector'[r][s][t] = vector'[r][s][t] + range;
if (vector'[r][s][t] > high)
        vector'[r][s][t] = vector'[r][s][t] - range;

if ( (mv_format == "field") && (t==1) && (VOP_structure == "Frame VOP") )
        PMV[r][s][t] = vector'[r][s][t] * 2;
else
        PMV[r][s][t] = vector'[r][s][t];
```

The parameters in the bitstream shall be such that the reconstructed differential motion vector, *delta*, shall lie in the range [*low*:*high*]. In addition the reconstructed motion vector, *vector'*[*r*][*s*][*t*], and the updated value of the motion vector predictor *PMV*[*r*][*s*][*t*], shall also lie in the range [*low* : *high*]. The allowed range [*low* : *high*] for the motion vectors depends on the parameter vop_fcode[s][t].

r_size, *f*, *delta*, *high*, *low* and *range* are temporary variables that are not used in the remainder of this specification.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

motion_code[r][s][t] and motion_residual[r][s][t] are fields recovered from the bitstream. mv_format is recovered from the bitstream using Table 6-102 and Table 6-102.

r, *s* and *t* specify the particular motion vector component being processed as identified in Table 7-38.

vector[*r*][*s*][*t*] is the final reconstructed motion vector for the luminance component of the macroblock.

7.16.7.4.2 Motion vector restrictions

In frame VOPs, the vertical component of the field motion vectors shall be restricted so that they only cover half the range that is supported by the vop_fcode that relates to those motion vectors. This restriction ensures that the motion vector predictors will always have values that are appropriate for decoding subsequent frame motion vectors. Table 7-39 summarises the size of motion vectors that may be coded as a function of vop_fcode.

Table 7-39 — Allowable motion vector range as a function of *vop_fcode*[*s*][*t*]

***vop_fcode*[*s*][*t*]**	**Vertical components (*t*==1) of field vectors in frame VOPs**	**All other cases**
0	(forbidden)	
1	[-4: +3,5]	[-8: +7,5]
2	[-8: +7,5]	[-16: +15,5]
3	[-16: +15,5]	[-32: +31,5]
4	[-32: +31,5]	[-64: +63,5]
5	[-64: +63,5]	[-128: +127,5]
6	[-128: +127,5]	[-256: +255,5]
7	[-256: +255,5]	[-512: +511,5]
8	[-512: +511,5]	[-1024: +1023,5]
9	[-1024: +1023,5]	[-2048: +2047,5]
10-14	(reserved)	
15	(used when a particular *vop_fcode*[*s*][*t*] will not be used)	

7.16.7.4.3 Updating motion vector predictors

Once all of the motion vectors present in the macroblock have been decoded using the process defined in the previous clause it is sometimes necessary to update other motion vector predictors. This is because in some prediction modes fewer than the maximum possible number of motion vectors are used. The remainder of the predictors that might be used in the VOP must retain “sensible” values in case they are subsequently used.

The motion vector predictors shall be updated as specified in Table 7-40 and Table 7-41. The rules for updating motion vector predictors in the case of skipped macroblocks are specified in 7.16.5.7.

NOTE It is possible for an implementation to optimise the updating (and resetting) of motion vector predictors depending on the VOP type. For example in a P-VOP the predictors for backwards motion vectors are unused and need not be maintained.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-40 — Updating of motion vector predictors in frame VOPs

frame_motion_-type	macroblock_motion_-forward	macroblock_-intra	Predictors to Update
Frame-based‡	-	1	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Frame-based	1	0	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Frame-based‡	0	0	*PMV*[*r*][*s*][*t*] = 0 §
Field-based	1	0	(none)

NOTE - *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that;
PMV[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

‡ **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based

§ (Only occurs in P-VOP) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.16.7.4.4.

Table 7-41 — Updating of motion vector predictors in field VOPs

field_motion_-type	macroblock_motion_-forward	macroblock_-intra	Predictors to Update
Field-based‡	-	1	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Field-based	1	0	*PMV*[1][0][1:0] = *PMV*[0][0][1:0]
Field-based‡	0	0	*PMV*[*r*][*s*][*t*] = 0 §
16x8 MC	1	0	(none)

NOTE - *PMV*[*r*][*s*][1:0] = *PMV*[*u*][*v*][1:0] means that;
PMV[*r*][*s*][1] = *PMV*[*u*][*v*][1] and *PMV*[*r*][*s*][0] = *PMV*[*u*][*v*][0]

‡ **field_motion_type** is not present in the bitstream but is assumed to be Field-based

§ (Only occurs in P-VOP) *PMV*[*r*][*s*][*t*] is set to zero (for all *r*, *s* and *t*). See 7.16.7.4.4.

7.16.7.4.4 Resetting motion vector predictors

All motion vector predictors shall be reset to zero in the following cases:

- At the start of each slice.
- Whenever a macroblock is completely transparent.
- Whenever an intra macroblock is decoded.
- Whenever a macroblock is encoded as DPCM residuals.
- In a P-VOP when a macroblock is skipped (not coded = 1).

7.16.7.4.5 Prediction in P-VOP

In the case that a P field VOP is used as the second field of a frame in which the first field is an I field VOP a series of semantic restrictions apply. These ensure that prediction is only made from the I field VOP. These restrictions are;

- Field prediction in which **motion_vertical_field_select** indicates the same parity as the field being predicted shall not be used.
- There shall be no skipped macroblocks (not_coded = 0).

7.16.7.4.6 Motion vectors for chrominance components

The motion vectors calculated in the previous clauses refer to the luminance component where;

$$vector[r][s][t] = vector'[r][s][t] \quad \text{(for all r, s and t)}$$

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

For each of the two chrominance components the motion vectors shall be scaled as follows:

4:2:0 Both the horizontal and vertical components of the motion vector are scaled by dividing by two:

vector[*r*][*s*][0] = *vector*'[*r*][*s*][0] / 2;
vector[*r*][*s*][1] = *vector*'[*r*][*s*][1] / 2;

4:2:2 The horizontal component of the motion vector is scaled by dividing by two, the vertical component is not altered:

vector[*r*][*s*][0] = *vector*'[*r*][*s*][0] / 2;
vector[*r*][*s*][1] = *vector*'[r][s][1];

4:4:4 The motion vector is unmodified:

vector[*r*][*s*][0] = *vector*'[*r*][*s*][0];
vector[*r*][*s*][1] = *vector*'[*r*][*s*][1];

7.16.7.4.7 Semantic restrictions concerning predictions

It is a restriction on the bitstream that reconstructed motion vectors shall not refer to samples outside the decoded area of a reference VOP. For an arbitrary shape VOP, the decoded area refers to the area within the bounding rectangle, padded as described in subclause 7.16.7.1.1. A bounding rectangle is defined by vop_width and vop_height extended to multiple of 16.

7.16.7.5 Forming predictions

Predictions are formed by reading prediction samples from the reference fields or frames. A given sample is predicted by reading the corresponding sample in the reference field or frame offset by the motion vector in the absolute coordinate system.

A positive value of the horizontal component of a motion vector indicates that the prediction is made from samples (in the reference field/frame) that lie to the right of the samples being predicted in the absolute coordinate system.

A positive value of the vertical component of a motion vector indicates that the prediction is made from samples (in the reference field/frame) that lie the below the samples being predicted in the absolute coordinate system.

All motion vectors are specified to an accuracy of one half sample. Thus if a component of the motion vector is odd, the samples will be read from mid-way between the actual samples in the reference field/frame. These half-samples are calculated by simple linear interpolation from the actual samples.

In the case of field-based predictions it is necessary to determine which of the two available fields to use to form the prediction. In the case of field-based prediction and 16x8 MC an additional bit, motion_vertical_field_select, is encoded to indicate which field to use.

If motion_vertical_field_select is zero then the prediction is taken from the top reference field.

If motion_vertical_field_select is one then the prediction is taken from the bottom reference field.

For each prediction block the integer sample motion vectors int_vec[t] and the half sample flags half_flag[t] shall be formed as follows;

-`-`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
for (t=0 ; t<2 ; t++) {
        int_vec[t] = vector[r][s][t] //// 2 ;
        if ((vector[r][s][t] – (2 * int_vec[t]) != 0)
            half_flag[t] = 1 ;
        else
            half_flag[t] = 0 ;
}
```

Then for each sample in the prediction block the samples are read and the half sample prediction applied as follows;

```
if ( ( ! half_flag[0] )&& ( ! half_flag[1]) )
    pel_pred[y][x] = pel_ref[y + int_vec[1]][x + int_vec[0]] ;

if ( ( ! half_flag[0] )&& half_flag[1] )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]+1][x + int_vec[0]] ) // 2 ;

if ( half_flag[0]&& ( ! half_flag[1]) )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]][x + int_vec[0]+1] ) // 2 ;

if ( half_flag[0]&& half_flag[1] )
    pel_pred[y][x] = ( pel_ref[y + int_vec[1]][x + int_vec[0]] +
                       pel_ref[y + int_vec[1]][x + int_vec[0]+1] +
                           pel_ref[y + int_vec[1]+1][x + int_vec[0]] +
                               pel_ref[y + int_vec[1]+1][x + int_vec[0]+1] ) // 4 ;
```

where *pel_pred*[*y*][*x*] is the prediction sample being formed and *pel_ref*[*y*][*x*] are samples in the reference field or frame.

7.16.7.6 Motion vector selection

Table 7-42 shows the prediction modes used in field VOPs and Table 7-43 shows the predictions used in frame VOPs. In each table the motion vectors that are present in the bitstream are listed in the order in which they appear in the bitstream.

Table 7-42 — Predictions and motion vectors in field VOPs

field_ motion_ type	macroblock_motion_- forward	macro- block_- intra	Motion vector	Prediction formed for
Field-based‡	-	1	*vector*'[0][0][1:0]	None
Field-based	1	0	*vector*'[0][0][1:0]	whole field, forward
Field-based‡	0	0	*vector*'[0][0][1:0]*§	whole field, forward
16x8 MC	1	0	*vector*'[0][0][1:0] *vector*'[1][0][1:0]	upper 16x8 field, forward lower 16x8 field, forward

NOTE - Motion vectors are listed in the order they appear in the bitstream

‡ **field_motion_type** is not present in the bitstream but is assumed to be Field-based

* These motion vectors are not present in the bitstream

§ The motion vector is taken to be (0; 0) as explained in 7.16.7.4.5.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 7-43 — Predictions and motion vectors in frame VOPs

frame_-motion_-type	macroblock_motion_-forward	macro-block_-intra	Motion vector	Prediction formed for
Frame-based‡	-	1	*vector*'[0][0][1:0]	None
Frame-based	1	0	*vector*'[0][0][1:0]	frame, forward
Frame-based‡	0	0	*vector*'[0][0][1:0]*§	frame, forward
Field-based	1	0	*vector*'[0][0][1:0] *vector*'[1][0][1:0]	top field, forward bottom field, forward

NOTE - Motion vectors are listed in the order they appear in the bitstream

‡ **frame_motion_type** is not present in the bitstream but is assumed to be Frame-based

* These motion vectors are not present in the bitstream

§ The motion vector is taken to be (0; 0) as explained in 7.16.7.4.5.

7.16.7.7 Skipped macroblocks

A skipped macroblock is a macroblock for which no data is encoded, that is part of a coded slice. With the exception of the first non-transparent macroblock in a slice, if not_coded flag in a macroblock is '1', the macroblock is to be skipped. The decoder shall form a prediction for skipped macroblocks which shall then be used as the final decoded sample values.

The process differs between field VOPs and frame VOPs.

There shall be no skipped macroblocks in I-VOPs.

7.16.7.7.1 P field VOP

- The prediction shall be made as if field_motion_type is "Field-based"
- The prediction shall be made from the field of the same parity as the field being predicted.
- Motion vector predictors shall be reset to zero.
- The motion vector shall be zero.

7.16.7.7.2 P frame VOP

- The prediction shall be made as if frame_motion_type is "Frame-based"
- Motion vector predictors shall be reset to zero.
- The motion vector shall be zero.

7.16.7.8 Combining predictions

The final stage is to combine the various predictions together in order to form the final prediction blocks.

It is also necessary to organise the data into blocks that are either field organised or frame organised in order to be added directly to the decoded coefficients.

The transform data is either field organised or frame organised as specified by *dct_type*.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.7.8.1 Simple frame predictions

In the case of simple frame predictions no further processing is required.

The predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 8 lines, 8 samples by 16 lines and 16 samples by 16 lines respectively.

7.16.7.8.2 Simple field predictions

In the case of simple field predictions (i.e. not 16×8) no further processing is required

In the case of simple field prediction in a frame picture the predictions for chrominance components of 4:2:0, 4:2:2 and 4:4:4 formats for each field shall be of size 8 samples by 4 lines, 8 samples by 8 lines and 16 samples by 8 lines respectively.

7.16.7.8.3 16x8 Motion compensation

In this prediction mode separate predictions are formed for the upper 16x8 region of the macroblock and the lower 16x8 region of the macroblock.

The predictions for chrominance components, for each 16x8 region, of 4:2:0, 4:2:2 and 4:4:4 formats shall be of size 8 samples by 4 lines, 8 samples by 8 lines and 16 samples by 8 lines respectively.

7.16.8 Output of the decoding process

This section describes the output of the theoretical model of the decoding process that decodes bitstreams conforming to this specification.

The decoding process input is one or more coded video bitstreams (one for each of the layers). The video layers are generally multiplexed by the means of a system stream that also contains timing information.

The output of the decoding process is a series of fields or frames that are normally the input of a display process. The order in which fields or frames are output by the decoding process is called the display order, and may be different from the coded order (when B-pictures are used). The display process is responsible for the action of displaying the decoded fields or frames on a display device. If the display device cannot display at the frame rate indicated in the bitstream, the display process may perform frame rate conversion. This specification does not describe a theoretical model of the display process nor the operation of the display process.

Since some of the syntax elements, such as progressive_frame, may be needed by the display process, in this theoretical model of the decoding process, all the syntactic elements that are decoded by the decoding process are output by the decoding process and may be accessed by the display process.

When a progressive sequence is decoded (progressive_sequence is equal to 1), the luminance and chrominance samples of the reconstructed frames are output by decoding process in the form of progressive frames and the output rate is the frame rate. Figure 7-116 illustrates this in the case of chroma_format equals to 4:2:0.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

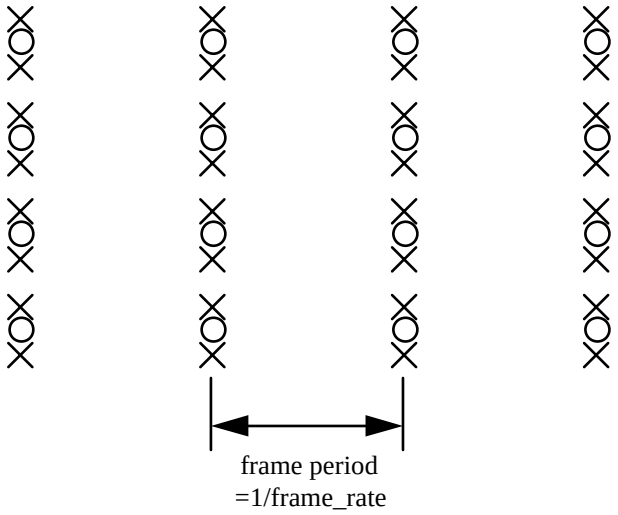


Figure 7-116 — progressive_sequence == 1

The same reconstructed frame is output one time if repeat_first_field is equal to 0, and two or three consecutive times if repeat_first_field is equal to 1, depending on the value of top_field_first. Figure 7-117 illustrates this in the case of chroma_format equals to 4:2:0 and repeat_first_field equals 1.

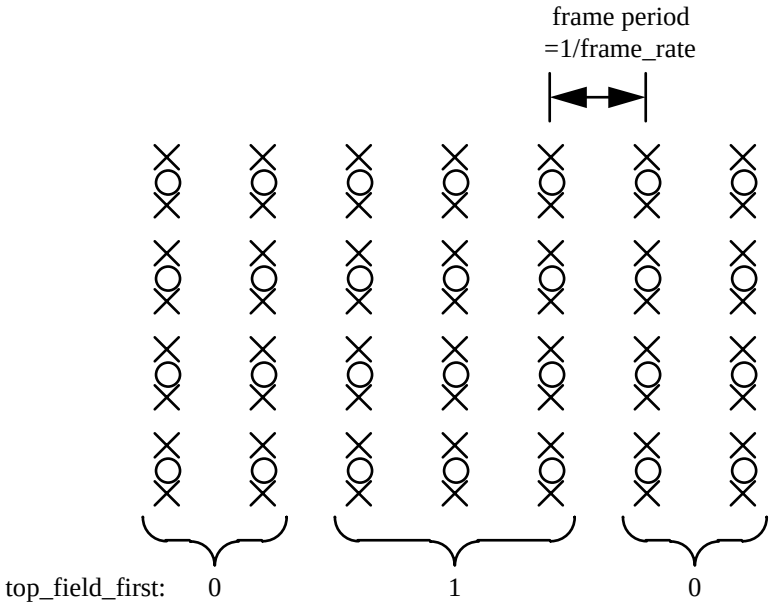


Figure 7-117 — progressive_sequence == 1, repeat_first_field = 1

When decoding an interlaced sequence (progressive_sequence is equal to 0), the luminance samples of the reconstructed frames are output by the decoding process in the form of interlaced fields at a rate that is twice the frame rate. Figure 7-118 illustrates this.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

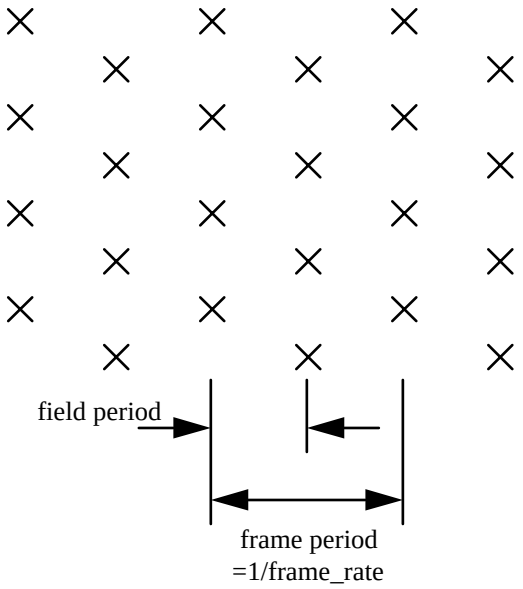


Figure 7-118 — progressive_sequence == 0

It is a requirement on the bitstream that the fields at the output of the decoding process shall always be alternately top and bottom (note that the very first field of a sequence may be either top or bottom).

If the reconstructed frame is interlaced (progressive_frame is equal to 0), the luminance samples and chrominance samples are output by the decoding process in the form of two consecutive fields. The first field output by the decoding process is the top field or the bottom field of the reconstructed frame, depending on the value of top_field_first.

Although all the samples of progressive frames represent the same instant in time, all the samples are not output at the same time by the decoding process when the sequence is interlaced.

If the reconstructed frame is progressive (progressive_frame is equal to 1), the luminance samples are output by the decoding process in the form of two or three consecutive fields, depending on the value of repeat_first_field.

NOTE The information that these fields originate from the same progressive frame in the bitstream is conveyed to the display process.

All of the chrominance samples of the reconstructed progressive frame are output by the decoding process at the same time as the first field of luminance samples. This is illustrated in Figure 7-119 and Figure 7-120.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

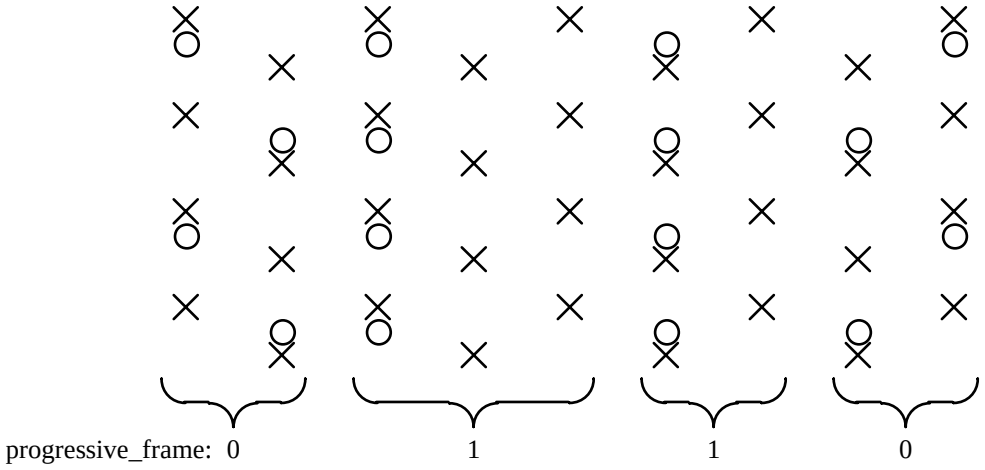


Figure 7-119 — progressive_sequence == 0 with 4:2:0 chrominance.

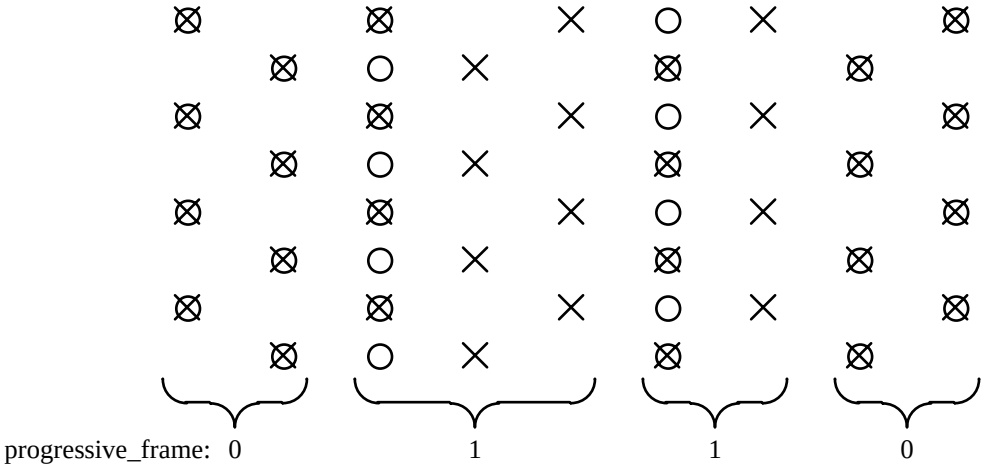


Figure 7-120 — progressive_sequence == 0 with 4:2:2 or 4:4:4 chrominance.

7.16.9 Sprite decoding

The subclause specifies the additional decoding process for a sprite video object. The sprite decoding can operate in two modes: basic sprite decoding and low-latency sprite decoding. Figure 7-121 is a diagram of the sprite decoding process. It is simplified for clarity.

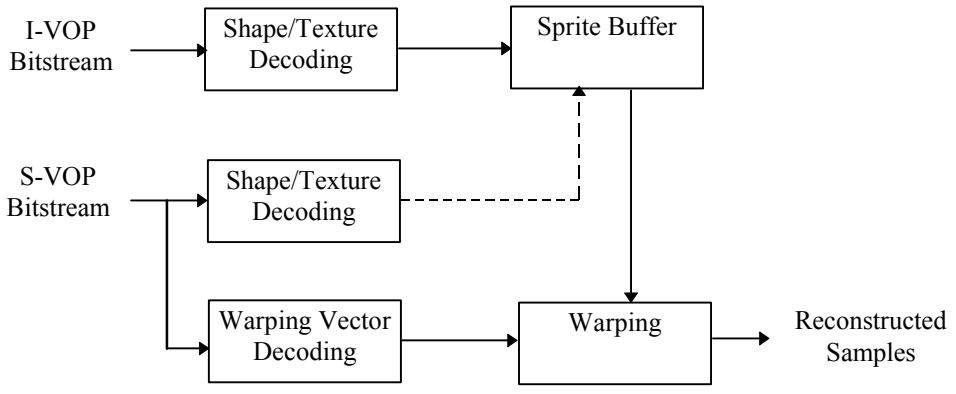


Figure 7-121 — The sprite decoding process

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.9.1 Higher syntactic structures

The various parameters in the VOL and VOP bitstreams shall be interpreted as described in Clause 6. When sprite_enable == '1', vop_coding_type shall be "I" only for the initial VOP in a VOL for basic sprites (i.e. low_latency_sprite_enable == '0'), and all the other VOPs shall be S-VOPs (i.e. vop_coding_type == "S"). The reconstructed I-VOP in a VOL for basic sprites is not displayed but stored in a sprite memory, and will be used by all the remaining S-VOPs in the same VOL. An S-VOP is reconstructed by applying warping to the VOP stored in the sprite memory, using the warping parameters (i.e. a set of motion vectors) embedded in the VOP bitstream. Alternatively, in a VOL for low-latency sprites (i.e. low_latency_sprite_enable == '1'), these S-VOPs can update the information stored in the sprite memory before applying warping.

7.16.9.2 Sprite Reconstruction

The luminance, chrominance and grayscale alpha data of a sprite are stored in two-dimensional arrays. The width and height of the luminance array are specified by sprite_width and sprite_height respectively. The samples in the sprite luminance, chrominance and grayscale alpha arrays are addressed by two-dimensional integer pairs (i', j') and (i_c', j_c') as defined in the following:

- Top left luminance and grayscale alpha sample
  (i', j') = (sprite_left_coordinate, sprite_top_coordinate)

- Bottom right luminance and grayscale alpha sample
  (i', j') = (sprite_left_coordinate + sprite_width − 1,
  sprite_top_coordinate + sprite_height − 1)

- Top left chrominance sample
  For 4:2:0 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2, sprite_top_coordinate / 2)
  For 4:2:2 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2, sprite_top_coordinate)
  For 4:4:4 VOPs,
  (i_c', j_c') = (sprite_left_coordinate, sprite_top_coordinate)

- Bottom right chrominance sample
  For 4:2:0 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2 + sprite_width// 2 − 1,
  sprite_top_coordinate / 2 + sprite_height// 2 − 1).
  For 4:2:2 VOPs,
  (i_c', j_c') = (sprite_left_coordinate / 2 + sprite_width// 2 − 1,
  sprite_top_coordinate + sprite_height − 1).
  For 4:4:4 VOPs,
  (i_c', j_c') = (sprite_left_coordinate + sprite_width − 1,
  sprite_top_coordinate + sprite_height − 1).

- Likewise, the addresses of the luminance, chrominance and grayscale alpha samples of the VOP currently being decoded are defined in the following:

- Top left sample of luminance and grayscale alpha
  (i, j) = (0, 0) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) for non-rectangular VOPs

- Bottom right sample of luminance and grayscale alpha
  (i, j) = (video_object_layer_width - 1, video_object_layer_height - 1) for rectangular VOPs, and
  (i, j) = (vop_horizontal_mc_spatial_ref + vop_width - 1,
  vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular VOPs

- Top left sample of chrominance
  (i_c, j_c) = (0, 0) for rectangular VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2, vop_vertica_mc_spatial_ref / 2) for non-rectangular 4:2:0 VOPs

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

(i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2, vop_vertical_mc_spatial_ref) for non-rectangular 4:2:2 VOPs
(i_c, j_c) = (vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) for non-rectangular 4:4:4 VOPs

- Bottom right sample of chrominance
  (i_c, j_c) = (video_object_layer_width / 2 - 1, video_object_layer_height / 2 - 1) for rectangular 4:2:0 VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2 + vop_width// 2 - 1,
  vop_vertical_mc_spatial_ref / 2 + vop_height// 2 - 1) for non-rectangular 4:2:0 VOPs
  (i_c, j_c) = (video_object_layer_width / 2 - 1, video_object_layer_height - 1) for rectangular 4:2:2 VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref / 2 + vop_width// 2 - 1,
  vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular 4:2:2 VOPs
  (i_c, j_c) = (video_object_layer_width - 1, video_object_layer_height - 1) for rectangular 4:4:4 VOPs, and
  (i_c, j_c) = (vop_horizontal_mc_spatial_ref + vop_width- 1,
  vop_vertical_mc_spatial_ref + vop_height - 1) for non-rectangular 4:4:4 VOPs

7.16.9.3 Sprite reference point decoding

The syntactic elements in sprite_trajectory () and below shall be interpreted as specified in Clause 6. du[i] and dv[i] ($0 \le i < $ no_sprite_point) specifies the mapping between indexes of some reference points in the VOP and the corresponding reference points in the sprite. These points are referred to as VOP reference points and sprite reference points, respectively, in the rest of the specification.

The index values for the VOP reference points are defined as:

(i_0, j_0) = (0, 0) when video_object_layer_shape == 'rectanglar', and
(vop_horizontal_mc_spatial_ref, vop_vertical_mc_spatial_ref) otherwise,
(i_1, j_1) = (i_0+W, j_0),
(i_2, j_2) = (i_0, j_0 + H),
(i_3, j_3) = (i_0+W, j_0+H)

where W = video_object_layer_width and H = video_object_layer_height when video_object_layer_shape == 'rectanglar' or W = vop_width and H = vop_height otherwise. Only the index values with subscripts less than no_sprite_point shall be used for the rest of the decoding process.

The index values for the sprite reference points shall be calculated as follows:

(i_0', j_0') = (s / 2) (2 i_0 + du[0], 2 j_0 + dv[0])
(i_1', j_1') = (s / 2) (2 i_1 + du[1] + du[0], 2 j_1 + dv[1] + dv[0])
(i_2', j_2') = (s / 2) (2 i_2 + du[2] + du[0], 2 j_2 + dv[2] + dv[0])
(i_3', j_3') = (s / 2) (2 i_3 + du[3] + du[2] + du[1] + du[0], 2 j_3 + dv[3] + dv[2] + dv[1] + dv[0])

where i_0', j_0', etc are integers in $\frac{1}{s}$ pel accuracy, where s is specified by sprite_warping_accuracy. Only the index values with substcripts less than no_sprite_point need to be calculated.

When no_of_sprite_warping_points == 2 or 3, the index values for the *virtual sprite points* are additionally calculated as follows:

(i_1'', j_1'') = (16 (i_0 + W') + ((W – W') (r i_0' – 16 i_0) + W' (r i_1' – 16 i_1)) // W,
16 j_0 + ((W – W') (r j_0' – 16 j_0) + W' (r j_1' – 16 j_1)) // W)
(i_2'', j_2'') = (16 i_0 + ((H – H') (r i_0' – 16 i_0) + H' (r i_2' – 16 i_2)) // H,
16 (j_0 + H') + ((H – H') (r j_0' – 16 j_0) + H' (r j_2' – 16 j_2)) // H)

where i_1'', j_1'', i_2'', and j_2'' are integers in $\frac{1}{16}$ pel accuracy, and r = 16/s. W' and H' are defined as the smallest integers that satisfy the following condition:

$W' = 2^{\alpha}$, $H' = 2^{\beta}$, $W' \ge W$, $H' \ge H$, $\alpha > 0$, $\beta > 0$, both α and β are integers.

The calculation of i_2'', and j_2'' is not necessary when no_of_sprite_warping_points == 2.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.16.9.4 Warping

For any pixel (i, j) inside the VOP boundary, ($F(i, j)$, $G(i, j)$) and ($F_c(i_c, j_c)$, $G_c(i_c, j_c)$) are computed as described as follows. These quantities are then used for sample reconstruction as specified in subclause 7.16.9.5. The following notations are used to simplify the description:

$I = i - i_0$,
$J = j - j_0$,

For 4:2:0 VOPs,

$I_c = 4\ i_c - 2\ i_0 + 1$,

$J_c = 4\ j_c - 2\ j_0 + 1$,

For 4:2:2 VOPs,

$I_c = 4\ i_c - 2\ i_0 + 1$, for calculating $F_c(i_c, j_c)$,

$I_c = 2\ i_c - i_0$, for calculating $G_c(i_c, j_c)$,

$J_c = 2\ j_c - 2\ j_0 + 1$, for calculating $F_c(i_c, j_c)$,

$J_c = j_c - j_0$, for calculating $G_c(i_c, j_c)$,

For 4:4:4 VOPs,

$I_c = i_c - i_0$,

$J_c = j_c - j_0$,

When no_of_sprite_warping_point == 0,

$(F(i, j), G(i, j)) = (si, sj)$,
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (si_c, sj_c)$.

When no_of_sprite_warping_point == 1,

$(F(i, j), G(i, j)) = (i_0' + sI, j_0' + sJ)$,
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' /// 2 + s\ (i_c - i_0 / 2),\ j_0' /// 2 + s\ (j_c - j_0 / 2))$, for 4:2:0 VOPs.

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' /// 2 + s\ (i_c - i_0 / 2),\ j_0' + sJ_c)$, for 4:2:2 VOPs.

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' + sI_c, j_0' + sJ_c)$, for 4:4:4 VOPs.

When no_of_sprite_warping_points == 2,

$(F(i, j), G(i, j)) = (\ i_0' + ((-r\ i_0' + i_1'')\ I + (r\ j_0' - j_1'')\ J) /// (W'\ r),$
$j_0' + ((-r\ j_0' + j_1'')\ I + (-r\ i_0' + i_1'')\ J) /// (W'\ r))$,
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\ i_0' + i_1'')\ I_c + (r\ j_0' - j_1'')\ J_c + 2\ W'\ r\ i_0' - 16W') /// (4\ W'\ r),$
$((-r\ j_0' + j_1'')\ I_c + (-r\ i_0' + i_1'')\ J_c + 2\ W'\ r\ j_0' - 16W') /// (4\ W'\ r))$,

for 4:2:0 VOPs,

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W') /// (4\,W'\,r),$
$j_0' + ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c) /// (W'\,r))$,

for 4:2:2 VOPs,

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (\,i_0' + ((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c) /// (W'\,r),$
$j_0' + ((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c) /// (W'\,r))$,

for 4:4:4 VOPs.

According to the definition of W' and H' (i.e. $W' = 2^{\alpha}$ and $H' = 2^{\beta}$), the divisions by "///" in these functions can be replaced by binary shift operations. By this replacement, the above equations can be rewritten as:

$(F(i, j), G(i, j)) = (\,i_0' + (((-r\,i_0' + i_1'')\,I + (r\,j_0' - j_1'')\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),$
$j_0' + (((-r\,j_0' + j_1'')\,I + (-r\,i_0' + i_1'')\,J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),$
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),$
$((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2\,W'\,r\,j_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2))$,

for 4:2:0 VOPs,

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2\,W'\,r\,i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),$
$j_0' + (((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2^{\alpha+\rho-1}))$,

for 4:2:2 VOPs,

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (\,i_0' + (((-r\,i_0' + i_1'')\,I_c + (r\,j_0' - j_1'')\,J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),$
$j_0' + (((-r\,j_0' + j_1'')\,I_c + (-r\,i_0' + i_1'')\,J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho))$,

for 4:4:4 VOPs.

where $2^{\rho}=r$.

When no_of_sprite_warping_points == 3,

$(F(i, j), G(i, j)) = (i_0' + ((-r\,i_0' + i_1'')\,H'\,I + (-r\,i_0' + i_2'')W'\,J) /// (W'H'r),$
$j_0' + ((-r\,j_0' + j_1'')\,H'\,I + (-r\,j_0' + j_2'')W'\,J) /// (W'H'r))$,
$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,H'\,I_c + (-r\,i_0' + i_2'')W'\,J_c + 2\,W'H'r\,i_0' - 16W'H') /// (4W'H'r),$
$((-r\,j_0' + j_1'')\,H'\,I_c + (-r\,j_0' + j_2'')W'\,J_c + 2\,W'H'r\,j_0' - 16W'H') /// (4W'H'r))$,

for 4:2:0 VOPs,

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\,i_0' + i_1'')\,H'\,I_c + (-r\,i_0' + i_2'')W'\,J_c + 2\,W'H'r\,i_0' - 16W'H') /// (4W'H'r),$
$j_0' + ((-r\,j_0' + j_1'')\,H'\,I_c + (-r\,j_0' + j_2'')W'\,J_c) /// (W'H'r))$

for 4:2:2 VOPs,

$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' + ((-r\,i_0' + i_1'')\,H'\,I_c + (-r\,i_0' + i_2'')W'\,J_c) /// (W'H'r),$
$j_0' + ((-r\,j_0' + j_1'')\,H'\,I_c + (-r\,j_0' + j_2'')W'\,J_c) /// (W'H'r))$,

for 4:4:4 VOPs.

According to the definition of W' and H', the computation of these functions can be simplified by dividing the denominator and numerator of the division beforehand by W' (when $W' < H'$) or H' (when $W' \geq H'$). As in the case of no_of_sprite_warping_points == 2, the divisions by "///" in these functions can be replaced by binary shift operations. For example, when $W' \geq H'$ (i.e. $\alpha \geq \beta$) the above equations can be rewritten as:

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$(F(i, j), G(i, j)) = (i_0' + (((-r\, i_0' + i_1'')\, I + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta}\, J + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),$$
$$j_0' + (((-r\, j_0' + j_1'')\, I + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta}\, J + 2^{\alpha+\rho-1}) >> (\alpha+\rho))),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\, i_0' + i_1'')\, I_c + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta}\, J_c + 2W'r\, i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),$$
$$((-r\, j_0' + j_1'')\, I_c + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta}\, J_c + 2W'r\, j_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2)),$$

for 4:2:0 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (((-r\, i_0' + i_1'')\, I_c + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta}\, J_c + 2W'r\, i_0' - 16W' + 2^{\alpha+\rho+1}) >> (\alpha+\rho+2),$$
$$j_0' + (((-r\, j_0' + j_1'')\, I_c + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta}\, J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho))),$$

for 4:2:2 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = (i_0' + (((-r\, i_0' + i_1'')\, I_c + (-r\, i_0' + i_2'')\, 2^{\alpha-\beta}\, J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho)),$$
$$j_0' + (((-r\, j_0' + j_1'')\, I_c + (-r\, j_0' + j_2'')\, 2^{\alpha-\beta}\, J_c + 2^{\alpha+\rho-1}) >> (\alpha+\rho))),$$

for 4:4:4 VOPs.

When no_of_sprite_warping_point == 4,

$$(F(i, j), G(i, j)) = ((a\, I + b\, J + c) /// (g\, I + h\, J + D\, W\, H),$$
$$(d\, I + e\, J + f) /// (g\, I + h\, J + D\, W\, H)),$$
$$(F_c(i_c, j_c), G_c(i_c, j_c)) = ((2\, a\, I_c + 2\, b\, J_c + 4\, c - (g\, I_c + h\, J_c + 2\, D\, W\, H)\, s) /// (4\, g\, I_c + 4\, h\, J_c + 8\, D\, W\, H),$$
$$(2\, d\, I_c + 2\, e\, J_c + 4\, f - (g\, I_c + h\, J_c + 2\, D\, W\, H)\, s) /// (4\, g\, I_c + 4\, h\, J_c + 8\, D\, W\, H)),$$

for 4:2:0 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = ((2\, a\, I_c + 2\, b\, J_c + 4\, c - (g\, I_c + h\, J_c + 2\, D\, W\, H)\, s) /// (4\, g\, I_c + 4\, h\, J_c + 8\, D\, W\, H),$$
$$(d\, I_c + e\, J_c + f) /// (g\, I_c + h\, J_c + D\, W\, H)),$$

for 4:2:2 VOPs,

$$(F_c(i_c, j_c), G_c(i_c, j_c)) = ((a\, I_c + b\, J_c + c) /// (g\, I_c + h\, J_c + D\, W\, H),$$
$$(d\, I_c + e\, J_c + f) /// (g\, I_c + h\, J_c + D\, W\, H)),$$

for 4:4:4 VOPs.

where

$$g = ((i_0' - i_1' - i_2' + i_3')\,(j_2' - j_3') - (i_2' - i_3')\,(j_0' - j_1' - j_2' + j_3'))\, H,$$
$$h = ((i_1' - i_3')\,(j_0' - j_1' - j_2' + j_3') - (i_0' - i_1' - i_2' + i_3')\,(j_1' - j_3'))\, W,$$
$$D = (i_1' - i_3')\,(j_2' - j_3') - (i_2' - i_3')\,(j_1' - j_3'),$$
$$a = D\,(i_1' - i_0')\, H + g\, i_1',$$
$$b = D\,(i_2' - i_0')\, W + h\, i_2',$$
$$c = D\, i_0'\, W\, H,$$
$$d = D\,(j_1' - j_0')\, H + g\, j_1',$$
$$e = D\,(j_2' - j_0')\, W + h\, j_2',$$
$$f = D\, j_0'\, W\, H.$$

A set of parameters that causes the denominator of any of the above equations to be zero for any pixel in an opaque or boundary macroblock is disallowed. The implementor should be aware that a 32 bit register may not be sufficient for representing the denominator or the numerator in the above transform functions for affine and perspective transforms. The usage of a 64 bit floating point representation should be sufficient in such cases.

7.16.9.5 Sample reconstruction

The reconstructed value Y of the luminance sample (i, j) in the currently decoded VOP shall be defined as

$$Y = ((s - r_j)((s - r_i)\, Y_{00} + r_i\, Y_{01}) + r_j\,((s - r_i)\, Y_{10} + r_i\, Y_{11}))\, //\, s^2,$$

-`-`-,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

where Y_{00}, Y_{01}, Y_{10}, Y_{11} represent the sprite luminance sample at ($F(i, j)$////s, $G(i, j)$////s), ($F(i, j)$////s + 1,$G(i, j)$////s), ($F(i, j)$////s, $G(i, j)$////s + 1), and ($F(i, j)$////s + 1,$G(i, j)$////s + 1) respectively, and $r_i = F(i, j) - (F(i, j)////s)s$ and $r_j = G(i, j) - (G(i, j)////s)s$. Figure 7-121 illustrates this process.

In case any of Y_{00}, Y_{01}, Y_{10} and Y_{11} lies outside the sprite luminance binary mask, it shall be obtained by the padding process as defined in subclause 7.16.7.1.1.

When brightness_change_in_sprite == 1, the final reconstructed luminance sample (i, j) is further computed as Y = Y * (brightness_change_factor + 100)) // 100, clipped to the range of [0, 2^(bits_per_pixel-1)].

Similarly, the reconstructed value C of the chrominance sample (i_c, j_c) in the currently decoded VOP shall be defined as

$$C = ((s - r_j)((s - r_i) C_{00} + r_i C_{01}) + r_j ((s - r_i) C_{10} + r_i C_{11})) // s^2,$$

where C_{00}, C_{01}, C_{10}, C_{11} represent the sprite chrominance sample at ($F_c(i_c, j_c)$////s, $G_c(i_c, j_c)$////s), ($F_c(i_c, j_c)$////s + 1, $G_c(i_c, j_c)$////s), ($F_c(i_c, j_c)$////s, $G_c(i_c, j_c)$////s + 1), and ($F_c(i_c, j_c)$////s + 1, $G_c(i_c, j_c)$////s + 1) respectively, and $r_i = F_c(i_c, j_c) - (F_c(i_c, j_c))////s)s$ and $r_j = G_c(i_c, j_c) - (G_c(i_c, j_c)////s)s$. In case any of C_{00}, C_{01}, C_{10} and C_{11} lies outside the sprite chrominance binary mask, it shall be obtained by the padding process as defined in subclause 7.16.7.1.1.

The same method is used for the reconstruction of grayscale alpha and luminance samples. The reconstructed value A of the grayscale alpha sample (i, j) in the currently decoded VOP shall be defined as

$$A = ((s - r_j)((s - r_i) A_{00} + r_i A_{01}) + r_j ((s - r_i) A_{10} + r_i A_{11})) // s^2,$$

where A_{00}, A_{01}, A_{10}, A_{11} represent the sprite grayscale alpha sample at ($F(i, j)$////s, $G(i, j)$////s), ($F(i, j)$////s + 1,$G(i, j)$////s), ($F(i, j)$////s, $G(i, j)$////s + 1), and ($F(i, j)$////s + 1,$G(i, j)$////s + 1) respectively, and $r_i = F(i, j) - (F(i, j)////s)s$ and $r_j = G(i, j) - (G(i, j)////s)s$. In case any of A_{00}, A_{01}, A_{10} and A_{11} lies outside the sprite luminance binary mask, it shall be obtained by the padding process as defined in subclause 7.16.7.1.1.

The reconstructed values of luminance binary mask samples $BY(i,j)$ shall be computed following the identical process for the luminance samples. However, corresponding binary mask sample values shall be used in place of luminance samples Y_{00}, Y_{01}, Y_{10}, Y_{11}. Assume the binary mask sample opaque is equal to 255 and the binary mask sample transparent is equal to 0. If the computed value is bigger or equal to 128, $BY(i, j)$ is defined as opaque. Otherwise, BY (i, j) is defined as transparent. The chrominance binary mask samples shall be reconstructed by decimating the corresponding 2 x 2 adjacent luminance binary mask samples as specified in subclause 7.16.7.1.1.4.

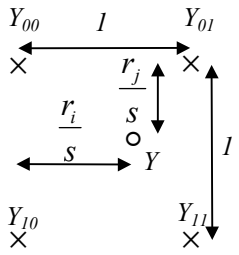


Figure 7-122 — Pixel value interpolation (it is assumed that sprite samples are located on an integer grid)

7.17 The FGS decoding process

This clause specifies the FGS decoding process that the decoder shall perform to recover visual data from the coded bitstream and reconstruct the enhancement VOP. In a typical application of FGS, the bitstream at the input of an FGS decoder is a truncated version of the bitstream at the output of an FGS encoder. It is likely that, at the end of each fgs vop bitstream before the next fgs_vop_start_code, only partial bits of an fgs block are at the input

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

of the decoder due to truncation of the fgs vop bitstream. These partial bits of an fgs block may be discarded. The description of the FGS decoding process in the following sub-sections is for an fgs block that contains all of its bits.

7.17.1 Bit-plane decoding of the absolute values of the DCT coefficients

The bit-plane decoding procedure can be described as follows:

- The absolute values of all the DCT coefficients are initialized to zeros.
- Variable length decoding of (RUN, EOP) symbols is performed. If frequency weighting is used, i.e., **fgs_frequency_weighting_enable** is '1', the choice of VLC tables for decoding depends on the first value of the frequency weighting matrix (shifted bits for the DC component). Table 7-44 indicates this dependency:

Table 7-44 — VLC Table Choices for Frequency Weighting

fw_matrix[0]	VLC tables
0	B.4.1 table_bcp
1	B.4.1 table_bcp
2	B.4.2 table_bcp4fw_m
3	B.4.2 table_bcp4fw_m
4	B.4.3 table_bcp4fw_h
5	B.4.3 table_bcp4fw_h
6	B.4.3 table_bcp4fw_h
7	B.4.3 table_bcp4fw_h

  where table_bpc, table_ bpc4fw_m, and table_bpc4fw_h are given in Annex B.4. Four 2-D VLC tables are used in each set. The first table in each set corresponds to the MSB block-bp. The second table in each set corresponds to the second MSB block-bp. The third table in each set corresponds to the third MSB block-bp. The fourth table in each set corresponds to the fourth MSB block-bp and all the lower block-bps. For the ESCAPE cases, RUN is represented with 6 bits to represent values from 0 to 63 and EOP is a 1-bit flag to indicate whether the end of the block-bp is reached (with value 1) or not (with value 0).

- The (RUN,EOP) symbols are translated into RUN number of consecutive 0's before a 1 and whether to fill 0's to the end of this block-bp. If the ALL-ZERO symbol is decoded, this block-bp contains all 0's. The ALL-ZERO symbol is not in the VLC tables for the MSB block-bp.
- Accumulate the bit to the partial result of each absolute value of the (bit shifted) DCT coefficients at the proper significant position.
- If frequency weighting or selective enhancement is used, the bit shifted DCT coefficient is right-shifted by S bits where S = fw_matrix[i]+ fgs_shifted_bits, and fw_matrix[i] is the ith element of the frequency weighting matrix.
- After decoding all the bits for the fgs vop available to the FGS decoder, partial absolute values of all the DCT coefficients are available.

Note that the MSB block-bp in the above description is defined as the first block-bp of an 8x8 DCT block that contains at least one "1" bit value. The MSB block-bp of an 8x8 DCT block may or may not be in the first vop-bp.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.17.2 Decoding of the signs of the DCT coefficients

A sign bit is decoded from the enhancement layer bitstream immediately after the (RUN, EOP) code corresponding to the MSB of the non-zero DCT coefficient.

7.17.3 Reconstruction of the enhancement DCT coefficients

Let DIFF and SIGN be the absolute value and sign of a DCT coefficient decoded from the enhancement bitstream, and ENH_COEFF be the reconstructed enhancement DCT coefficient.

ENH_COEFF = SIGN*DIFF

7.17.4 Reconstruction of the enhanced VOP

After enhanced DCT coefficients are obtained and inverse zigzag scan is used to convert the enhanced DCT coefficients into 8x8 blocks, an inverse DCT is applied to each 8x8 block. Inverse zigzag scan is performed according to Figure 7-4 (c) in subclause 7.4.2. If **fgs_vop_coding_type** is "I", the reconstructed residue VOP is added to the base-layer VOP pixel by pixel. As a reference to this operation, the base-layer VOP is the one after clipping. If **fgs_vop_coding_type** is "P" or "B", the reconstructed block after the inverse DCT followed by adding the motion compensated prediction block is the enhancement VOP. Motion vector decoding is performed according to subclause 7.6.3. Motion compensation is performed according to the applicable parts of subclause 7.9.1.1, subclause 7.9.1.3.2, and subclause 7.9.1.3.3. The reconstructed enhancement VOP pixels are limited into the value range between 0 and 255 by the clipping unit in the FGS enhancement layer to generate the final enhancement VOP.

8 Visual-Systems Composition Issues

8.1 Temporal Scalability Composition

Background composition is used in forming the background region for objects at the enhancement layer of temporal scalability when the value of both enhancement_type and background_composition is one. This process is useful when the enhancement VOP corresponds to the partial region of the VOP belonging to the reference layer. In this process, the background of a current enhancement VOP is composed using the previous and the next VOPs in display order belonging to the reference layer.

Figure 8-1 shows the background composition for the current frame at the enhancement layer. The dotted line represents the shape of the selected object at the previous VOP in the reference layer (called "forward shape"). As the object moves, its shape at the next VOP in the reference layer is represented by a broken line (called "backward shape").

For the region outside these shapes, the pixel value from the nearest VOP at the reference layer is used for the composed background. For the region occupied only by the forward shape, the pixel value from the next VOP at the reference layer is used for the composed frame. This area is shown as lightly shaded in Figure 8-1. On the other hand, for the region occupied only by the backward shape, pixel values from the previous VOP in the reference layer are used. This is the area shaded dark in Figure 8-1. For the region where the areas enclosed by these shapes overlap, the pixel value is given by padding from the surrounding area. The pixel value which is outside of the overlapped area should be filled before the padding operation.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

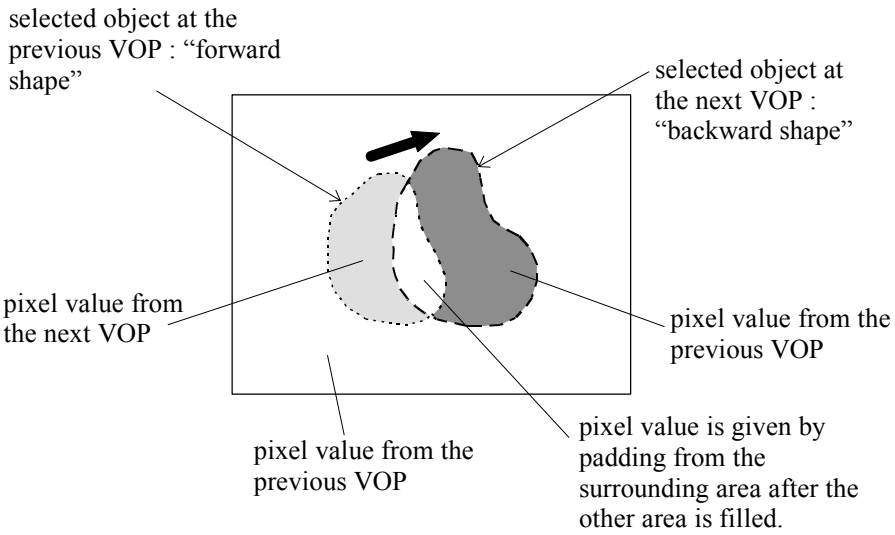


Figure 8-1 — Background composition

The following process is a mathematical description of the background composition method.

If s(x,y,ta)=0 and s(x,y,td)=0
fc(x,y,t) = f(x,y,td) (|t-ta|>|t-td|)
fc(x,y,t) = f(x,y,ta) (otherwise),
if s(x,y,ta)=1 and s(x,y,td)=0
fc(x,y,t) = f(x,y,td)
if s(x,y,ta)=0 and s(x,y,td)=1
fc(x,y,t) = f(x,y,ta)
if s(x,y,ta)=1 and s(x,y,td)=1
The pixel value of fc(x,y,t) is given by repetitive padding from the surrounding area.

where

fc composed background
f decoded VOP at the reference layer
s shape information (alpha plane) , 0: transparent, 1: opaque
(x,y) the spatial coordinate
t time of the current VOP
ta time of the previous VOP
td time of the next VOP

Two types of shape information, *s*(*x*, *y*, *ta*) and *s*(*x*, *y*, *td*), are necessary for the background composition. *s*(*x*, *y*, *ta*) is called a "forward shape" and *s*(*x*, *y*, *td*) is called a "backward shape". If f(*x, y, ta*) does not exist, the pixel value of fc(*x, y, t*) for s(*x, y, td*) = 0 is given by f(*x, y, td*) and the pixel value of fc(*x, y, t*) for s(*x, y, td*) = 1 is given by repetitive padding from the boundary pixel value. If f(*x, y, td*) does not exist, pixel value of fc(*x, y, t*) for s(*x, y, ta*) = 0 is given by f(*x, y, ta*) and the pixel value of fc(*x, y, t*) for s(*x, y, ta*) = 1 is given by repetitive padding from the boundary pixel value.

8.2 Sprite Composition

The static sprite technology enables to encode very efficiently video objects which content is expected not to vary in time along a video sequence. For example, it is particularly well suited to represent backgrounds of scenes (decor, landscapes) or logos.

A static sprite (sometimes referred as mosaic in the literature) is a frame containing spatial information for a single object, obtained by gathering information for this object throughout the sequence in which it appears. A static sprite can be a very large frame: it can correspond for instance to a wide angle view of a panorama.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

The ISO/IEC 14496-2 syntax defines a dedicated coding mode to obtain VOPs from static sprites: the so-called "S-VOPs". S-VOPs with sprite_enable == 'static' are extracted from a static sprite using a warping operation consisting in a global spatial transformation driven by few motion parameters (0,2,4, 6 or 8).

For composition with other VOPs, there are no special rules for S-VOPs with sprite_enable == 'static'. However, it is classical to use S-VOPs with sprite_enable == 'static' as background objects over which "classical" objects are superimposed.

8.3 Mesh Object Composition

A Mesh Object represents the geometry of a sequence of 2D triangular meshes. This data can be used along with separately coded image texture data to render texture-mapped images, e.g., by the composition process as defined in ISO/IEC 14496-1:2001. A Mesh Object stream may be contained in part of a BIFS animation stream, as defined in ISO/IEC 14496-1:2001. In terminals implementing mesh animation functionality using both ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496, the decoded mesh data is used to update the appropriate fields of a BIFS IndexedFaceSet2D node, defined in ISO/IEC 14496-1:2001, for composition purposes. In this case, the appropriate fields of the IndexedFaceSet2D BIFS node are updated as described in the following.

a.) The coordinates of the mesh points (vertices) are obtained from the output of the Mesh Object decoder. The Mesh Object uses a pixel-based local coordinate system with *x*-axis pointing to the right and *y*-axis pointing down. However, ISO/IEC 14496-1:2001 specifies a coordinate system with *y*-axis pointing up. Therefore, a simple coordinate transform shall be applied to the *y*-coordinates of mesh points to ensure the proper orientation of the object after composition. The *y*-coordinate y_n of a decoded mesh node point *n* shall be transformed as follows:

$$Y_n = - y_n ,$$

where Y_n is the *y*-coordinate of this mesh node point in the coordinate system as specified in ISO/IEC 14496-1:2001. The origin of this object is at the top-left point. The same transform shall be applied to the coordinates of node points of each Mesh Object Plane (MOP).

b.) The coordinate indices are the indices of the mesh points forming faces (triangles) obtained from the output of the Mesh Object decoder. All decoded faces are triangles. The topology of a Mesh Object is constant starting from an intra-coded MOP, throughout a sequence of predictive-coded MOPs (until the next intra-coded MOP); therefore, the coordinate indices shall be updated only for intra-coded MOPs.

c.) Texture coordinates for mapping textures onto the mesh geometry are computed from the decoded node point locations of an intra-coded Mesh Object Plane and its bounding rectangle. Let x_{min}, y_{min} and x_{max}, y_{max} define the bounding rectangle of all node points of an intra-coded MOP. Then the width *w* and height *h* of the texture map shall be:

$$w = \text{ceil}(x_{max}) - \text{floor}(x_{min}) ,$$

$$h = \text{ceil}(y_{max}) - \text{floor}(y_{min}) .$$

A texture coordinate pair (s_n, t_n) is computed for each node point $p_n = (x_n, y_n)$ as follows:

$$s_n = (x_n - \text{floor}(x_{min}))/w ,$$

$$t_n = 1.0 - (y_n - \text{floor}(y_{min}))/h .$$

The topology of a Mesh Object is constant starting from an intra-coded MOP, throughout a sequence of predictive-coded MOPs (until the next intra-coded MOP); therefore, the texture coordinates shall be updated only for intra-coded MOPs.

d.) The texture coordinate indices are identical to the coordinate indices.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.4 Spatial Scalability composition

Background composition is used in forming the background region for objects at the enhancement layer of spatial scalability when the value of both enhancement_type and background_composition is '1'. This process is useful when the enhancement VOP corresponds to the partial region of the VOP belonging to the reference layer. In this process, current enhancement VOP is composed with up-sampled VOP has same display time in the base layer as back ground image.

In object based spatial scalability, if enhancement_type is "1", use_ref_shape is "1" and background_composition is '1', reference layer should be up-sampled in the same process described in subclause 7.9.2.6 for background composition. If reference layer has binary shape information, it shall be up-sampled in the same process described in subclause 7.5.4.4. And for the region outside shape of the enhancement VOP, the both texture and shape pixels should be placed on the up-sampled VOP in the reference layer.

In object based spatial scalability, if enhancement_type is "1", use_ref_shape is "0" and background_composition is '1', reference layer shape and texture should be up-sampled in the same process described in subclause 7.5.4.4 and 7.9.2.6. And for the shape and texture outside the enhancement VOP, the shape and texture of the up-sampled VOP of the reference layer should be used. In this case, enhancement layer shape and texture are derived by composing the shapes and textures inside enhancement VOP and ouside enhancement VOP. The shape outside enhancement VOP is given by the collocated shape of the up-sampled VOP of the reference layer, and the texture outside enhancement VOP is given by the collocated texture of the up-sampled VOP of the reference layer.

9 Profiles and Levels

NOTE In this part of ISO/IEC 14496 the word "profile" is used as defined below. It should not be confused with other definitions of "profile" and in particular it does not have the meaning that is defined by ISO/IEC ISP.

Profiles and levels provide a means of defining subsets of the syntax and semantics of this part of ISO/IEC 14496 and thereby the decoder capabilities required to decode a particular bitstream. A profile is a defined sub-set of the entire bitstream syntax that is defined by this part of ISO/IEC 14496. A level is a defined set of constraints imposed on parameters in the bitstream. Conformance tests will be carried out against defined profiles at defined levels.

The purpose of defining conformance points in the form of profiles and levels is to facilitate bitstream interchange among different applications. Implementers of this part of ISO/IEC 14496 are encouraged to produce decoders and bitstreams which correspond to those defined conformance regions. The discretely defined profiles and levels are the means of bitstream interchange between applications of this part of ISO/IEC 14496.

In this clause the constrained parts of the defined profiles and levels are described. All syntactic elements and parameter values which are not explicitly constrained may take any of the possible values that are allowed by this part of ISO/IEC 14496. A decoder shall be deemed to be conformant to a given profile at a given level if it is able to properly decode all allowed values of all syntactic elements as specified by that profile at that level.

9.1 Visual Object Types

The following four tables list the tools included in each of the Object Types. Bitstreams that represent a particular object corresponding to an Object Type shall not use any of the tools for which the table does not have an 'X'.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 9-1 — Tools for Version 1 Visual Object Types

Visual Tools	Visual Object Types								
	Simple	**Core**	**Main**	**Simple Scalable**	**N-bit**	**Animated 2D Mesh**	**Basic Animated Texture**	**Still Scalable Texture**	**Simple Face**
Basic • I-VOP • P-VOP • AC/DC Prediction • 4-MV, Unrestricted MV	X	X	X	X	X	X			
Error resilience • GOB Resynchronization • Data Partitioning • Reversible VLC	X	X	X	X	X	X			
Short Header	X	X	X		X	X			
B-VOP		X	X	X	X	X			
P-VOP with OBMC (Texture)									
Method 1/Method 2 Quantisation		X	X		X	X			
P-VOP based temporal scalability • Rectangular • Arbitrary Shape		X	X		X	X			
Binary Shape		X	X		X	X	X		
Grey Shape			X						
Interlace			X						
Sprite			X						
Temporal Scalability (Rectangular)				X					
Spatial Scalability (Rectangular)				X					
N-Bit					X				
Scalable Still Texture						X	X	X	
2D Dynamic Mesh with uniform topology						X	X		
2D Dynamic Mesh with Delaunay topology						X			
Facial Animation Parameters									X

NOTE 1: ″Binary Shape Coding” includes constant alpha.

NOTE 2: The parameters are restricted as follows for the tool “P-VOP based temporal scalability”:
- ref_select_code shall be either ‘00’ or ‘01’.
- vop_coding_type of the enhancement layer VOP shall be either ‘00’ or ‘01’.
- vop_coding_type of the reference layer VOP shall be either ‘00’ or ‘01’.
- load_backward_shape shall be ‘0’ and background composition is not performed.

NOTE 3: An ‘X’ for the “Method 1/Method 2 Quantisation” indicates that both quantisation methods are supported by the Visual Object Type. Where there is no ‘X’ the only quantisation method supported is the Second Inverse Quantisation Method (subclause 7.4.4.2).

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 9-2 — Tools for Version 2 Visual Object Types

Visual Tools	Visual Object Types				
	Advanced Real Time Simple	**Advanced Coding Efficiency**	**Advanced Scalable Texture**	**Core Scalable**	**Simple FBA**
Basic • I-VOP • P-VOP • AC/DC Prediction • 4-MV, Unrestricted MV	X	X		X	
Error resilience • GOB Resynchronization • Data Partitioning • Reversible VLC	X	X		X	
Short Header	X	X		X	
B-VOP		X		X	
P-VOP with OBMC (Texture)					
Method 1/Method 2 Quantisation		X		X	
P-VOP based temporal scalability • Rectangular • Arbitrary Shape		X		X	
Binary Shape		X		X	
Grey Shape		X			
Interlace		X			
Sprite					
Temporal Scalability (Rectangular)				X	
Spatial Scalability (Rectangular)				X	
N-Bit					
Scalable Still Texture			X		
2D Dynamic Mesh with uniform topology					
2D Dynamic Mesh with Delaunay topology					
Facial Animation Parameters					X
Body_Animation_Parameters					X
Reduced Resolution VOP	X				
NEWPRED	X				
Global Motion Compensation		X			
Quarter-pel Motion Compensation		X			
SA-DCT		X			
Error Resilience for Visual Texture Coding			X		
Wavelet Tiling			X		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Scalable Shape Coding for Still Texture			X		
Object Based Temporal Scalability				X	
Object Based Spatial Scalability				X	

Table 9-3 — Tools for Studio Visual Object Types

	Visual Object Type	
Visual Tools	**Simple Studio**	**Core Studio**
Basic -Progressive/Interlaced -Frame/Field Structure -Slice Structure -Studio DPCM Block -Studio Binary Shape -Studio Grayscale Shape	X	X
I-VOP	X	X
P-VOP		X
Studio Sprite		X

NOTE 3: The allowed values of ‘chroma_format’ in StudioVideoObjectLayer() are defined in the level definition.

NOTE 4: The allowed values of ‘bits_per_pixel’ and ‘alpha_bits_per_pixel’ in StudioVideoObjectLayer() are defined in the level definition

Table 9-4 — Tools for FGS Visual Object Types

Visual Tools	**Visual Object Types**	
	Advanced Simple	**Fine Granularity Scalable**
I-VOP	X	X
P-VOP	X	X
B-VOP	X	X
DC Prediction	X	X
AC Prediction	X	X
4-MV, Unrestricted MV	X	X
GOB Resynchronization	X	X
Data Partitioning	X	X
Reversible VLC	X	X
Short Header	X	X
Method 1/Method 2 Quantisation	X	X
Interlace	X	X
Global Motion Compensation	X	
Quarter-pel Motion Compensation	X	
Fine Granularity Scalability		X
FGS Temporal Scalability		X

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE 1 — The interlace tools are not used for levels L0, L1, L2, and L3 of AS and FGS Profiles.

NOTE 2 — The Advanced Simple Object Type is the base-layer of the Fine Granularity Scalable Object Type. If FGS Temporal Scalability is used, B-VOP is not allowed in the base-layer. However, B-VOP is allowed in the base-layer when FGS Temporal Scalability is not used in the Fine Granularity Scalable Object Type.

NOTE 3 — The Advanced Simple Object Type is the base-layer of the Fine Granularity Scalable Object Type. If FGS or FGS Temporal Scalability is used, Short Header is not allowed in the base-layer. However, Short Header is allowed in the base-layer when FGS or FGS Temporal Scalability is not used in the Fine Granularity Scalable Object Type.

9.2 Visual Profiles

Decoders that conform to a Profile shall be able to decode all objects that comply to the Object Types for which the table lists an 'X'.

Table 9-5 — Version 1 Visual Profiles

	Object Types / Profiles	Simple	Core	Main	Simple Scalable	N-Bit	Animated 2D Mesh	Basic Animated Texture	Scalable Texture	Simple Face
1.	Simple	X								
2.	Simple Scaleable	X			X					
3.	Core	X	X							
4.	Main	X	X	X					X	
5.	N-Bit	X	X			X				
6.	Hybrid	X	X				X	X	X	X
7.	Basic Animated Texture							X	X	X
8.	Scaleable Texture								X	
9.	Simple FA									X

Table 9-6 — Version 2 Visual Profiles

	Object Types / Profiles	Simple	Simple Scalable	Core	Core Scalable	Advanced Real Time Simple	Advanced Coding Efficiency	Advanced Scable Texture	Simple FBA
V2-1.	Advanced Real Time Simple	X				X			
V2-2.	Core Scalable	X	X	X	X				
V2-3.	Advanced Coding Efficiency	X		X			X		
V2-4.	Advanced Core	X		X				X	
V2-5.	Advanced Scalable Texture							X	
V2-6.	Simple FBA								X

Note Object Types that are not listed in this table are not decodable by the decoders complying to the Profiles listed in this table.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 9-7 — Studio Visual Profiles

	Object Types Profiles	Simple Studio	Core Studio
AMD1-1	Simple Studio	X	
AMD1-2	Core Studio	X	X

Note that object types that are not listed in this table are not decordable by decoders complying to the Profiles listed in this table.

Note that the Profiles listed in this table can be grouped into Natural Visual.

Table 9-8 shows the definition of Advanced Simple Profile and Fine Granularity Scalable Profile.

Table 9-8 — Definition of Advanced Simple Profile and Fine Granularity Scalable Profile

ID	Object Types Profiles	Simple	Advanced Simple	Fine Granularity Scalable
AS	Advanced Simple	X	X	
FGS	Fine Granularity Scalable	X	X	X

9.3 Visual Profiles@Levels

9.3.1 Natural Visual

The table that describes the natural visual profiles is given in Annex N.

9.3.2 Synthetic Visual

9.3.2.1 Scalable Texture Profile

Table 9-9 — Scalable texture profile levels

Profile	Level	Default wavelet filter	Max. down-load filter length	Max. no. of decom-position levels	Typical visual session size[1]	Max. Qp value	Max. no. of pixels/ session	VCV decoder rate (equi-valent MB/s)[2]	Max. no. of bit-planes for DC values	Max. VCV buffer size (equivalent MB)[2]
Scalable Texture	L3	Float, Integer	ON, 15	10	8192 x8192	12 bits	67108864	262144	18	262144
Scalable Texture	L2	Integer	ON, 15	8	2048 x2048	10 bits	4194304	16384	16	16384
Scalable Texture	L1	Integer	OFF	5	704 x576	8 bits	405504	1584	13	1584

Note: (1) This column is for informative use only. It provides an example configuration of the max. no. of pixels/session.
(2) This still texture VCV model is separate from the global video VCV model. An equivalent MB corresponds to 256 pixels.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.2.2 Simple Face Animation Profile

All ISO/IEC 14496-2 facial animation decoders (for all object types) are required to generate at their output a facial model including all the feature points defined in this part of ISO/IEC 14496, even if some of the features points will not be affected by any information received from the encoder.

The Simple Face object is not required to implement the viseme_def/expression_def functionality.

Level 1:

- number of objects: 1,
- The total FAP decode frame-rate in the bitstream shall not exceed 72 Hz ,
- The decoder shall be capable of a face model rendering update of at least 15 Hz, and
- Maximum bitrate 16 kbit/s.

Level 2:

- maximum number of objects: 4,
- The FAP decode frame-rate in the bitstream shall not exceed 72 Hz (this means that the FAP decode framerate is to be shared among the objects),
- The decoder shall be capable of rendering the face models with the update rate of at least 60 Hz, sharable between faces, with the constraint that the update rate for each individual face is not required to exceed 30 Hz, and
- Maximum bitrate 32 kbit/s.

9.3.2.3 Simple FBA Profile

All ISO/IEC 14496-2 FBA decoders (for all object types) are required to generate at their output a humanoid model including all the feature points and joints defined in this part of ISO/IEC 14496, even if some of the features points and joints will not be affected by any information received from the encoder.

Level 1:

- number of objects: 1
- The total FBA decode frame-rate in the bitstream shall not exceed 72 Hz
- The decoder shall be capable of a humanoid model rendering update of at least 15 Hz
- Maximum bitrate 32 kbit/s
- The decoder is not required to animate Spine3, Spine4 and Spine5 BAP groups

Level 2:

- maximum number of objects: 4
- The FBA decode frame-rate in the bitstream shall not exceed 72 Hz (this means that the FBA decode framerate is to be shared among the objects)
- The decoder shall be capable of rendering the humanoid models with the update rate of at least 60 Hz, sharable between humanoids, with the constraint that the update rate for each individual humanoid is not required to exceed 30 Hz
- Maximum bitrate 64 kbit/s

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.2.4 Advanced Core Profile and Advanced Scalable Texture Profile

Table 9-10 — Advanced core profile levels

Visual Profile	Level	Default wavelet filter	Max. down-load filter, length	Max. no. of decom-position levels	Typical visual session size[1]	Max. Qp value (bits)	Max. no. of pixels / session[2]	VCV decoder rate (equi-valent MB/s)[3]	Max. no. of bit-plains for DC values	Max. VCV buffer size (equi-valent MB)	Max. STO packet length (bits)	Max. no. of pixels / tile	Max. no. of tile
Advanced Core	L2	Integer	ON, 15	8	8192 x8192	10	67108864	262144	16	262144	8192	262144	2048
Advanced Core	L1	Integer	OFF	5	2048 x2048	8	4194304	16384	13	16384	4096	65536	1024
Advanced Scalable Texture	L3	Float, Integer	ON, 15	10	8192 x8192	12	67108864	262144	18	262144	8192	67108864	4096
Advanced Scalable Texture	L2	Integer	ON, 15	8	2048 x2048	10	4194304	16384	16	16384	4096	4194304	2048
Advanced Scalable Texture	L1	Integer	OFF	5	704 x576	8	405504	1584	13	1584	2048	405504 (=4xCIF)	1024

Note: (1) This column is for informative use only. It provides an example configuration of the max. no. of pixels/session.
(2) When the number of pixels/session is larger than max. no. of pixels/tile, tiling_disable shall be 0.
(3) This still texture VCV model is separate from the global video VCV model. An equivalent MB corresponds to 256 pixels.

9.3.3 Synthetic/Natural Hybrid Visual

The *Levels* of the Profiles which support both Natural Visual Object Types and Synthetic Visual Object Types are specified by giving bounds for the natural objects and for the synthetic objects. Parameters like bitrate can be combined across natural and synthetic objects.

9.3.3.1 Basic Animated Texture Profile

Level 1 = Simple Facial Animation Profile @ Level 1 + Scalable Texture @ Level 1 + the following restrictions on Basic Animated Texture object types:

- Maximum number of Mesh objects (with uniform topology): 4,
- Maximum total number of nodes (vertices) in Mesh objects: 480,
  (= 4 x nr. Of nodes of a uniform mesh covering a QCIF image with 16x16 pixel elements),
- Maximum frame-rate of a Mesh object: 30 Hz, and
- Maximum bitrate of Mesh objects: 64 kbit/sec.

Level 2 = Simple Facial Animation Profile @ Level 2 + Scalable Texture @ Level 2 + the following restrictions on Basic Animated Texture object types:

- Maximum number of Mesh objects (with uniform topology): 8,
- Maximum total number of nodes (vertices) in Mesh objects: 1748,
  (= 4 x nr. of nodes of a uniform mesh covering a CIF image with 16x16 pixel elements),
- Maximum frame-rate of a Mesh object: 60 Hz, and
- Maximum bitrate of Mesh objects: 128 kbit/sec.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.3.2 Hybrid Profile

Level 1 = Core Visual Profile @ Level 1 + Basic Animated Texture Profile @ Level 1 + the following restrictions on Animated 2D Mesh object types:

- Maximum number of Mesh objects (with uniform or Delaunay topology): 4
  (= maximum number of objects in visual session)
- Maximum total number of nodes (vertices) in Mesh objects: 480
  (= 4 x nr. of nodes of a uniform mesh covering a QCIF image with 16x16 pixel elements)
- Maximum frame-rate of a Mesh object: 30 Hz
  (= maximum frame-rate of video object)
- Maximum bitrate of Mesh objects: 64 kbit/sec.

Level 2 = Core Visual Profile @ Level 2 + Basic Animated Texture Profile @ Level 2 + the following restrictions on Animated 2D Mesh object types:

- Maximum number of Mesh objects(with uniform or Delaunay topology): 8
  (= maximum number of objects in visual session)
- Maximum total number of nodes (vertices) in Mesh objects: 1748
  (= 4 x nr. of nodes of a uniform mesh covering a CIF image with 16x16 pixel elements)
- Maximum frame-rate of a Mesh object: 60 Hz
  (= 2 x the maximum frame-rate of video object)
- Maximum bitrate of Mesh objects: 128 kbit/sec.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(normative)

Coding transforms

A.1 Discrete cosine transform for video texture

The NxN two dimensional DCT is defined as:

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

with u, v, x, y = 0, 1, 2, … N-1

where x, y are spatial coordinates in the sample domain

u, v are coordinates in the transform domain

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u,v = 0 \\ 1 & \text{otherwise} \end{cases}$$

The inverse DCT (IDCT) is defined as:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

If each pixel is represented by *n* bits per pixel, the input to the forward transform and output from the inverse transform is represented with (*n*+1) bits. The coefficients are represented in (*n*+4) bits. The dynamic range of the DCT coefficients is [-2^{n+3}:$+2^{n+3}$-1].

The N by N inverse discrete transform shall conform to IEEE Standard Specification for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, Std 1180-1990, December 6, 1990, with the following modifications:

1) In item (1) of subclause 3.2 of the IEEE specification, the last sentence is replaced by: <<Data sets of 1 000 000 (one million) blocks each should be generated for (L=256, H=255), (L=H=5) and (L=384, H=383).>>

2) The text of subclause 3.3 of the IEEE specification is replaced by : <<For any pixel location, the peak error shall not exceed 2 in magnitude. There is no other accuracy requirement for this test.>>

3) Let F be the set of 4096 blocks Bi[y][x] (i=0..4095) defined as follows:

a) Bi[0][0] = i - 2048

b) Bi[7][7] = 1 if Bi[0][0] is even, Bi[7][7] = 0 if Bi[0][0] is odd

c) All other coefficients Bi[y][x] other than Bi[0][0] and Bi[7][7] are equal to 0

For each block Bi[y][x] that belongs to set F defined above, an IDCT that claims to be compliant shall output a block f[y][x] that as a peak error of 1 or less compared to the reference saturated mathematical integer-number IDCT fíí(x,y). In other words, | f[y][x] - fíí(x,y)| shall be <= 1 for all x and y.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE 1 Clause 2.3 Std 1180-1990 “Considerations of Specifying IDCT Mismatch Errors” requires the specification of periodic intra-picture coding in order to control the accumulation of mismatch errors. Every macroblock is required to be refreshed before it is coded 132 times as predictive macroblocks. Macroblocks in B-pictures (and skipped macroblocks in P-pictures) are excluded from the counting because they do not lead to the accumulation of mismatch errors. This requirement is the same as indicated in 1180-1990 for visual telephony according to ITU-T Recommendation H.261.

NOTE 2 Whilst the IEEE IDCT standard mentioned above is a necessary condition for the satisfactory implementation of the IDCT function it should be understood that this is not sufficient. In particular attention is drawn to the following sentence from subclause 5.4: “Where arithmetic precision is not specified, such as the calculation of the IDCT, the precision shall be sufficient so that significant errors do not occur in the final integer values.”

A.1.1 Discrete cosine transform for the Studio Profile

The NxN two dimensional DCT is defined as:

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

with u, v, x, y = 0, 1, 2, … N-1

where x, y are spatial coordinates in the sample domain

u, v are coordinates in the transform domain

$$C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } u,v = 0 \\ 1 & \text{otherwise} \end{cases}$$

The inverse DCT (IDCT) is defined as:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

If each pixel is represented by *n* bits per pixel, the input to the forward transform and output from the inverse transform is represented with (*n*+1) bits. The bit precision of the DCT coefficients changes in accordance with the value of mpeg2_stream defined in StudioVideoObjectLayer(). In case of mpeg2_stream = 0, The coefficients are represented in (n+7) bits including three fractional bits. The dynamic range of the coefficients is $[-2^{n+6}:+2^{n+6}-1]$. In case of mpeg2_stream = 1, The coefficients are represented in (n+4) bits. The dynamic range is $[-2^{n+3}:+2^{n+3}-1]$.

The N by N inverse discrete transform shall conform to IEEE Standard Specification for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, Std 1180-1990, December 6, 1990.

NOTE 1 Sublause 2.3 Std 1180-1990 “Considerations of Specifying IDCT Mismatch Errors” requires the specification of periodic intra-picture coding in order to control the accumulation of mismatch errors. Every macroblock is required to be refreshed before it is coded 132 times as predictive macroblocks. Macroblocks in B-pictures (and skipped macroblocks in P-pictures) are excluded from the counting because they do not lead to the accumulation of mismatch errors. This requirement is the same as indicated in 1180-1990 for visual telephony according to ITU-T Recommendation H.261.

NOTE 2 Whilst the IEEE IDCT standard mentioned above is a necessary condition for the satisfactory implementation of the IDCT function it should be understood that this is not sufficient. In particular, attention is drawn to the following sentence from subclause 5.4: “Where arithmetic precision is not specified, such as the calculation of the IDCT, the precision shall be sufficient so that significant errors do not occur in the final integer values.”

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.2 Discrete wavelet transform for still texture

A.2.1 Adding the mean

Before applying the inverse wavelet transform, the mean of each color component (“mean_y”, “mean_u”, and “mean_v”) is added to the all wavelet coefficients of DC subband.

A.2.2 Wavelet filter

A 2-D separable inverse wavelet transfrom is used to synthesize the still texture. The default wavelet composition is performed using Daubechies (9,3) tap biorthogonal filter bank. The inverse DWT is performed either in floating or integer operations depending on the field “wavelet_filter_type”, defined in the syntax.

The floating filter coefficients are:

Lowpass	**g[ ] =**	
[0.35355339059327	0.70710678118655	0.35355339059327]
Highpass	**h[ ] =**	
[0.03314563036812	0.06629126073624	-0.17677669529665
-0.41984465132952	0.99436891104360	-0.41984465132952
-0.17677669529665	0.06629126073624	0.03314563036812]

The integer filter coefficients are:

Lowpass	**g[ ] =**	
32	64	32
Highpass	**h[ ] =**	
3	6	-16
-38	90	-38
-16	6	3

The synthesis filtering operation is defined as follows:

$$y[n] = \sum_{i=-1}^{1} L[n+i]*g[i+1] \quad + \sum_{i=-4}^{4} H[n+i]*h[i+4]$$

where

- n = 0, 1, ... N-1, and N is the number of output points;
- L[2*i] = xl[i] and L[2*i+1] = 0 for i=0,1,...,N/2-1, and {xl[i]} are the N/2 input wavelet coefficients in the low-pass band;
- H[2*i+1] = xh[i] and H[2*i] = 0 for i=0,1,...,N/2-1, and {xh[i]} are the N/2 input wavelet coefficients in the high-pass band.

NOTE 1 the index range for h[] is from 0 to 8;

NOTE 2 the index range for g[] is from 0 to 2;

NOTE 3 the index range for L[] is from -1 to N;

NOTE 4 the index range for H[] is from -4 to N+3; and

NOTE 5 the values of L[] and H[] for indexes less than 0 or greater than N-1 are obtained by symmetric extension described in the following subclause.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the case of integer wavelet, the outputs at each composition level are scaled down with dividing by 8192 with rounding to the nearest integer.

A.2.3 Symmetric extension

A symmetric extension of the input wavelet coefficients is performed and the up-sampled and extended wavelet coefficients are generated. Note that the extension process shown below is an example when the extension is performed before up-sampling and that only the generated coefficients are specified. Two types of symmetric extensions are needed, both mirror the boundary pixels. Type A replicates the edge pixel and Type B does not replicate the edge pixel. This is illustrated in Figure A.1 and Figure A.2, where the edge pixel is indicated by z. The types of extension for the input data to the wavelet filters are shown in Table A.1.

Type A **…v w x y z** | z y x w v…

Type B **…...v w x y** | z y x w v…

Figure A.1 — Symmetrical extensions at leading boundary

Type A …v w x y z | **z y x w v…**

Type B ….v w x y z | **y x w v….**

Figure A.2 — Symmetrical extensions at the trailing boundary

Table A.1 — Extension method for the input data to the synthesis filters

	boundary	**Extension**
lowpass input xl[] to 3-tap filter g[]	leading trailing	TypeB TypeA
highpass input xh[] to 9-tap filter h[]	leading trailing	TypeA TypeB

The generated up-sampled and extended wavelet coefficients L[] and H[] are eventually specified as follows:

low-pass band :…0 L[2] 0 | **L[0] 0 L[2] 0 … L[N-4] 0 L[N-2] 0 |** L[N-2] 0 L[N-4] 0 …

high-pass band:… H[3] 0 H[1] | **0 H[1] 0 H[3] … 0 H[N-3] 0 H[N-1]** | 0 H[N-1] 0 H[N-3]…

A.2.4 Decomposition level

The number of decomposition levels of the luminance component is defined in the input bitstream. The number of decompostion levels for the chrominance components is one level less than that of the luminance components. If texture_object_layer width or texture_object_layer height cannot be divisible by (2 ^ decomposition_levels), then shape adaptive wavelet is applied.

A.2.5 Shape adaptive wavelet filtering and symmetric extension

A.2.5.1 Shape adaptive wavelet

The 2-D inverse shape adaptive wavelet transform uses the same wavelet filter as specified in Table A.1. According to the shape information, segments of consecutive output points are reconstructed and put into the correct locations. The filtering operation of shape adaptive wavelet is a generalization of that for the regular wavelet. The generalization allows the number of output points to be an odd number as well as an even number. Relative to the bounding rectangle, the starting point of the output is also allowed to be an odd number as well as an even number according to the shape information. Within the generalized wavelet filtering, the regular wavelet filtering is a special case where the number of output points is an even number and the starting point is an even

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

number (0) too. Another special case is for reconstruction of rectangular textures with an arbitrary size where the number of output points may be even or odd and the starting point is always even (0).

The same synthesis filtering is applied for shape-adaptive wavelet composition, i.e:

$$y[n] = \sum_{i=-1}^{1} L[n+i]*g[i+1] + \sum_{i=-4}^{4} H[n+i]*h[i+4]$$

where

- n = 0, 1, ... N-1, and N is the number of output points;
- L[2*i+s] = xl[i] and L[2*i+1-s] = 0 for i=0,1,...,(N+1-s)/2-1, and {xl[i]} are the (N+1-s)/2 input wavelet coefficients in the low-pass band;
- H[2*i+1-s] = xh[i] and H[2*i+s] = 0 for i=0,1,...,(N+s)/2-1, and {xh[i]} are the (N+s)/2 input wavelet coefficients in the high-pass band.

The only difference from the regular synthesis filtering is to introduce a binary parameter s in up-sampling, where s = 0 if the starting point of the output is an even number and s = 1 if the starting point of the output is an odd number.

The symmetric extension for the generalized synthesis filtering is specified in Table A.2 if N is an even number and in Table A.3 if N is an odd number.

Table A.2 — Extension method for the data to the synthesis wavelet filters if N is even

	Boundary	extension (s=0)	extension(s=1)
lowpass input xl[] to 3-tap filter g[]	Leading Trailing	TypeB TypeA	TypeA TypeB
highpass input xh[] to 9-tap filter h[]	Leading Trailing	TypeA TypeB	TypeB TypeA

Table A.3 — Extension method for the data to the synthesis wavelet filters if N is odd

	Boundary	extension(s=0)	extension(s=1)
lowpass input xl[] to 3-tap filter g[]	Leading Trailing	TypeB TypeB	TypeA TypeA
highpass input xh[] to 9-tap filter h[]	Leading Trailing	TypeA TypeA	TypeB TypeB

A.3 Shape-Adaptive DCT (SA-DCT)

Similar to standard 8x8-DCT from subclause A.1, SA-DCT converts a 8x8-block with texture data f[y][x] into a block with corresponding SA-DCT coefficients and, vice versa, inverse SA-DCT converts inverse quantised coefficients F[v][u] into decoded texture data f[x][y]. The definition of forward SA-DCT is based on the four subsequent processing steps S1 to S4 which are described in more detail in subclause A.3.1 on the basis of pseudo-C code.

The SA-DCT process is controlled by a binary array ‘f_shape[y][x]’ which describes the shape of input data f[y][x] and which is collocated with the corresponding 8x8-block. The elements of ‘f_shape[y][x]’ have the value ‘255’ for opaque and ‘0’ for transparent pels. This binary array has to be derived from the decoded BAB. For luminance blocks the elements of ‘f_shape[y][x]’ are taken from the corresponding subblock of 16x16-BAB and for chrominance blocks BAB is subsampled as specified for shape decoding.

An auxiliary binary array ‘shift_shape[y][x]’ and two one-dimensional counter arrays ‘pels_height[x]’ and ‘coeff_width[v]’ are implicitly computed from ‘f_shape[y][x]’ during the internal shift operations of forward SA-DCT (see S1 and S3). These auxiliary shape parameters are also needed for inverse SA-DCT. At the decoder, these

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

required shape parameters are first derived from decoded BAB during a pre-processing step I-S1 followed by four subsequent processing steps for inverse SA-DCT. The pre-processing step I-S1 as well as the four processing steps I-S2 to I-S5 of inverse SA-DCT are described in more detail in subclause A.3.2 on the basis of pseudo-C code.

NOTES -

1 The auxiliary array 'shift_shape[y][x]' describes the binary shape of the texture pattern 'shift_texture[y][x]' resulting from vertical shift of SA-DCT (see S1). The elements of 'pels_height[x]' denote the number of opaque pels in each column of f[y][x]. The elements of 'coeff_width[v]' describe the number SA-DCT coefficients existing in each row of F[v][u]. The permitted values of elements in 'pels_height[x]' and 'coeff_width[v]' lie in a range from 0 to 7.

2 The kernels of the one-dimensional DCT transforms used separately in horizontal and vertical direction (see steps S2 and S4) as well as the kernels of the corresponding one-dimensional inverse DCT transforms (see steps I-S2 and I-S4) have to be determined adaptively in dependence on the above mentioned shape parameters 'pels_height[x]' and 'coeff_width[v]'.

3 Floating point arithmetic should be used during entire forward and inverse SA-DCT (i.e. from S1 to S4 and from I-S2 to I-S5). The input to forward SA-DCT is an integer with 9 bits representing the dynamic range of [-256;+255]. The output of inverse SA-DCT is rounded to corresponding integers. The coefficients at the input of inverse SA-DCT are integers of 12 bits with a dynamic range of [-2048;+2047].

4 The arithmetic precision for the internal calculations of inverse SA-DCT, especially the one used during one-dimensional inverse DCT transform in processing steps I-S2 and I-S4 as well as the one of the intermediate arrays 'shift_intermediate[v][x]' and 'F_intermediate[v][x]', is not specified. Taking into account all possible shapes for 8x8-blocks with 'opaque_pels < 64', the precision shall be sufficient so that significant errors do not occur in the final integer values derived from f[y][x]. Analogous to inverse standard 8x8-DCT, the maximal error should meet the requirements of 1189-1990 for visual telephony.

A.3.1 Definition of Forward SA-DCT

S1: vertical shift of all opaque pels in f[y][x] to the most upper position and construction of corresponding auxiliary arrays 'shift_texture[y][x]' and 'shift_shape[y][x]

```
for (x=0; x<8; x++) {
   for (y=0; y<8; y++)
      shift_shape[y][x]=0;
   pels_count=0;
   for (y=0; y<8; y++)
      if(f_shape[y][x]==255) {
         shift_texture[pels_count][x]= f[y][x];
         shift_shape[pels_count][x]=255;
         pels_count++;
      }
   pels_height[x]=pels_count;
}
```

S2: application of shape-adaptive one-dimensional DCT to the first 'pels_height[x]' pels in all columns of 'shift_texture[y][x] with 'pels_height[x] != 0'

```
for (x=0; x<8; x++) {
   if(pels_height[x]==0) continue;
   scaling = sqrt(2.0/pels_height[x]);
   for (v=0; v<pels_height[x]; v++) {
      c0=1.0;
      if(v==0) c0= sqrt(0.5);
      F_intermediate[v][x]=0.0;
      for (y=0; y<pels_height[x]; y++){
         DCT_N = c0 * cos(v*(y+0.5)*(PI/pels_height[x]));
         F_intermediate[v][x] += scaling * DCT_N * shift_texture[y][x];
      }
   }
}
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

S3: horizontal shift of all intermediate coefficients in 'F_intermediate[v][x]' to most left position and construction of corresponding auxiliary array 'shift_intermediate[v][x]'

```
for (v=0; v<8; v++) {
   coeff_count=0;
   for (x=0; x<8; x++)
      if(shift_shape[v][x]==255) {
         shift_intermediate[v][coeff_count]= F_intermediate[v][x];
         coeff_count++;
      }
   coeff_width[v]=coeff_count
}
```

S4: application of shape-adaptive one-dimensional DCT to the first 'coeff_width[v]' intermediate coefficients in all rows of 'shift_intermediate[v][x]' with 'coeff_width[v] != 0'

```
for (v=0; (v<8) && (coeff_width[v]!=0); v++) {
   scaling = sqrt(2.0/coeff_width[v]);
   for (u=0; u<coeff_width[v]; u++) {
      c0=1.0;
      if(u==0) c0= sqrt(0.5);
      F[v][u]=0.0;
      for (x=0; x<coeff_width[v]; x++) {
         DCT_N = c0 * cos(u*(x+0.5)*(PI/coeff_width[v]));
         F[v][u] += scaling * DCT_N *shift_intermediate[v][x];
      }
   }
}
```

A.3.2 Definition of Inverse SA-DCT

I-S1: computation of required shape parameters 'coeff_width[v]', 'shift_shape[y][x]' and 'pels_height[x]' from decoded binary shape array 'f_shape[y][x]' (only needed at decoder)

```
opaque_pels=0;
for (x=0; x<8; x++) {
   for (y=0; y<8; y++)
      shift_shape[y][x]=0;
   pels_count=0;
   for (y=0; y<8; y++)
      if(f_shape[y][x]==255) {
         shift_shape[pels_count][x]=255;
         pels_count++;
      }
   pels_height[x]=pels_count
opaque_pels+=pels_count
}
for (v=0; v<8; v++) {
   coeff_count=0;
   for (x=0; x<8; x++)
      if(shift_shape[v][x]==255)
         coeff_count++;
   coeff_width[v]=coeff_count
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

I-S2: application of shape-adaptive inverse one-dimensional DCT to the first 'coeff_width[v]' decoded SA-DCT coefficients in all rows of 'F[v][u]' with 'coeff_width[v] != 0':

```
for (v=0; (v<8) && (coeff_width[v]!=0); v++) {
   scaling = sqrt(2.0/coeff_width[v]);
   for (x=0; x<coeff_width[v]; x++) {
      shift_intermediate[v][x] = 0.0;
      for (u=0; u<coeff_width[v]; u++) {
      c0=1.0;
      if(u==0) c0= sqrt(0.5);
         DCT_N = c0 * cos(u*(x+0.5)*(PI/coeff_width[v]));
         shift_intermediate[v][x] += scaling * DCT_N * F[v][u];
      }
   }
}
```

I-S3: re-shift of all reconstructed intermediate coefficients in 'shift_intermediate[v][x]' from most left position to their original column position defined by 'shift_shape[y][x]'

```
for (v=0; v<8; v++) {
   coeff_count=0;
   for (x=0; x<8; x++)
   if(shift_shape[v][x]==255) {
      F_intermediate[v][x]=shift_intermediate[v][coff_count];
      coeff_count++;
   }
}
```

I-S4: application of shape-adaptive inverse one-dimensional DCT to the first 'pel_height[x]' intermediate coefficients in all columns of 'F_intermediate[v][x] with 'pels_height[x] != 0'

```
for (x=0; x<8; x++) {
   if(pels_height[x]==0) continue;
   scaling = sqrt(2.0/pels_height[x]);
   for (y=0; y<pels_height[x]; y++) {
      shift_texture[y][x]=0.0;
      for (v=0; v<pels_height[x]; v++) {
         c0=1.0;
         if(v==0) c0= sqrt(0.5);
         DCT_N = c0 * cos(v*(y+0.5)*(PI/pels_height[x]));
         shift_texture[y][x] += scaling * DCT_N * F_intermediate[v][x];
      }
   }
}
```

I-S5: re-shift of all reconstructed pels in 'shift_texture[y][x]' from most upper position to their original row position defined by 'f_shape[y][x]'

```
for (x=0; x<8; x++) {
   pels_count=0;
   for (y=0; y<8; y++)
   if(f_shape[y][x]==255) {
      f[y][x]=shift_texture[x];
      pels_count++;
   }
}
```

A.4 SA-DCT with DC Separation and ΔDC Correction (ΔDC-SA-DCT)

ΔDC-SA-DCT is an extension of SA-DCT towards additional pre- and post-processing steps, called DC separation which is applied at the encoder before forward SADCT and ΔDC correction which is used at the decoder after

--`,,-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2004 – All rights reserved
Not for Resale

inverse SA-DCT. The definition of forward ΔDC-SA-DCT is based on three processing steps ΔS1 to ΔS3 combined with the four steps S1 to S4 of forward SA-DCT from subclause A.3.1. Steps ΔS1 to ΔS3 are described in more detail in subclause A.4.1 on the basis of pseudo-C code. Inverse ΔDC-SA-DCT is based on corresponding reverse processing steps I-ΔS1, I-ΔS2 and I-ΔS4 combined with I-S1 to I-S5 of inverse SA-DCT from subclause A.3.2 plus an additional step I-ΔS3 which corrects an systematical error caused by the fact that an eventually non-zero output coefficient F[0][0] of SA-DCT is not transmitted in ΔDC-SA-DCT (see step ΔS3). Subclause A.4.2 presents a description of steps I-ΔS1 to I-ΔS4 on basis of pseudo-C codes.

NOTES -

1 Floating point arithmetic should be used during entire forward and inverse ΔDC-SA-DCT (i.e. from ΔS1 to ΔS3 including S1 to S4 of SA-DCT and from I-ΔS1 to I-ΔS4 including I-S2 to I-S5 of inverse SA-DCT). The input to forward ΔDC-SA-DCT is an integer with 9 bits representing the dynamic range from [-256;+255]. The output of inverse ΔDC-SA-DCT is rounded to corresponding integers. The coefficients at the input of inverse ΔDC-SA-DCT are integers of 12 bits with a dynamic range from [-2048;+2047].

2 The arithmetic precision for the internal calculations of inverse ΔDC-SA-DCT, especially the one used for mean value calculation in I-ΔS1, for inverse SA-DCT in I-ΔS2, for calculation of the ΔDC correction term in I-ΔS3 and for final additions in I-ΔS4 as well as the one of the intermediate variables 'mean_value', 'corr_term', 'f[y][x]' and 'F[v][u]', is not specified. Taking into account all possible shapes for 8x8-blocks with 'opaque_pels < 64', the precision shall be sufficient so that significant errors do not occur in the final integer values of f[y][x]. Analogous to inverse standard DCT, the maximal error should meet the requirements of 1189-1990 for visual telephony.

A.4.1 Definition of Forward ΔDC-SA-DCT

ΔS1: calculation of mean value and subtraction of mean value from f[y][x]

```
mean_value=0.0;
mean_count=0;
for (y=0; y<8; y++)
   for (x=0; x<8; x++) {
      bin_shape[y][x]=0;
      if(f_shape[y][x]==255)
          bin_shape[y][x]=1;
      mean_value+=bin_shape[y][x]*f[y][x];
      mean_count+=bin_shape[y][x];
   }
mean_value *= 1.0/mean_count;
for (y=0; y<8; y++)
   for (x=0; x<8; x++)
      if (f_shape[y][x]==255)
      f[y][x]-= mean_value;
```

ΔS2: application of SA-DCT including processing steps S1 to S4 from subclause A.3.1 to zero-mean output data f[y][x] from processing step ΔS1

```
f[y][x] => S1 => S2 => S3 => S4 => F[v][u]
```

ΔS3: transmission of scaled mean value as coefficient F[0][0] (an eventually non-zero output coefficient [0][0] of SA-DCT in ΔS2 is overwritten and not transmitted in ΔDC-SA-DCT because it is redundant and can be reconstructed from other data received at the decoder)

```
F[0][0] = 8.0*mean_value
```

A.4.2 Definition of Inverse ΔDC-SA-DCT

I-ΔS1: extraction of re-scaled mean value from decoded coefficients F[v][u] and zero-setting of F[0][0]

```
mean_value = F[0][0]/8.0
F[0][0] = 0.0
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

I-ΔS2: application of inverse SA-DCT including processing steps I-S1 to I-S5 from subclause A.3.2 to modified output coefficients F[v][u] from processing step I-ΔS1 (I-S1 is only needed at the decoder)

```
F[v][u] => (I-S1) => I-S2 => I-S3 => I-S4 => => I-S5 => f[y][x]
```

I-ΔS3: calculation of ΔDC correction term

```
check_sum=0.0;
sqrt_sum=0.0;
for (x=0;x<8; x++) {
if(pels_height[x]==0) continue;
   sqrt_sum += sqrt(pels_height(x));
   for (y=0; y<8; y++)
      check_sum += bin_shape[y][x]*f[y][x];
}
corr_term = check_sum/sqrt_sum
```

I-ΔS4: addition of re-scaled mean value from step I-ΔS1 and weighted ΔDC correction term from step I-ΔS3 to output of inverse SA-DCT from step I-ΔS2

```
for (x=0; x<8; x++)
   if(pels_height[x]==0) continue;
   for (y=0; y<0; y++)
      if(bin_shape[y][x])
         f[y][x] += mean_value-(corr_term/sqrt(pels_height[x]));
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(normative)

Variable length codes and arithmetic decoding

B.1 Variable length codes

B.1.1 Macroblock type

Table B.1 — Macroblock types and included data elements for I- and P-, and S-VOPs

VOP type	mb type	Name	not_coded	mcbpc	mcsel	cbpy	dquant	mvd	mvd_{2-4}
P	not coded	-	1						
P	0	inter	1	1		1		1	
P	1	inter+q	1	1		1	1	1	
P	2	inter4v	1	1		1		1	1
P	3	intra	1	1		1			
P	4	intra+q	1	1		1	1		
P	stuffing	-	1	1					
I	3	intra		1		1			
I	4	intra+q		1		1	1		
I	stuffing	-		1					
S (update)	not_coded	-	1						
S (update)	0	inter	1	1		1			
S (update)	1	inter+q	1	1		1	1		
S (update)	3	intra	1	1		1			
S (update)	4	intra+q	1	1		1	1		
S (update)	stuffing	-	1	1					
S (piece)	3	intra		1		1			
S (piece)	4	intra+q		1		1	1		
S (piece)	stuffing	-		1					
S (GMC)	not coded	-	1						
S (GMC)	0	inter	1	1	1	1		1	
S (GMC)	1	inter+q	1	1	1	1	1	1	
S(GMC)	2	inter4v	1	1		1		1	1
S (GMC)	3	intra	1	1		1			
S (GMC)	4	intra+q	1	1		1	1		
S (GMC)	stuffing	-	1	1					

NOTE "1" means that the item is present in the macroblock
S (piece) indicates S-VOPs with low_latency_sprite_enable == 1 and sprite_transmit_mode == "piece"
S (update) indicates S-VOPs with low_latency_sprite_enable == 1 and sprite_transmit_mode == "update"
S (GMC) indicates S-VOPs with sprite_enable == "GMC"

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.2 — Macroblock types and included data elements for a P-VOP (scalability && ref_select_code == '11')

VOP Type	mb_type	Name	not_coded	mcbpc	cbpy	dquant	MVD	MVD_{2-4}
P	not coded	-	1					
P	0	INTER	1	1	1			
P	1	INTER+Q	1	1	1	1		
P	3	INTRA	1	1	1			
P	4	INTRA+Q	1	1	1	1		
P	stuffing	-	1	1				
NOTE "1" means that the item is present in the macroblock								

Table B.3 — VLC table for modb

Code	cbpb	mb_type
1		
01		1
00	1	1

Table B.4 — mb_type and included data elements in coded macroblocks in B-VOPs (ref_select_code != '00'||scalability=='0')

Code	dbquant	mvd_f	mvd_b	mvdb	mb_type
1				1	direct
01	1	1	1		interpolate mc+q
001	1		1		backward mc+q
0001	1	1			forward mc+q

Table B.5 — mb_type and included data elements in coded macroblocks in B-VOPs (ref_select_code == '00'&&scalability!='0')

Code	dbquant	mvd_f	mvd_b	mb_type
01	1	1		interpolate mc+q
001	1			backward mc+q
1	1	1		forward mc+q

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

B.1.2 Macroblock pattern

Table B.6 — VLC table for mcbpc for I-VOPs and S-VOPs with low_latence_sprite_enable==1 and sprite_transmit_mode==”piece”

Code	mbtype	cbpc (56)
1	3	00
001	3	01
010	3	10
011	3	11
0001	4	00
0000 01	4	01
0000 10	4	10
0000 11	4	11
0000 0000 1	Stuffing	--

Table B.7 — VLC table for mcbpc for P-VOPs and S-VOPs with low_latence_sprite_enable==1 and sprite_transmit_mode==”update”, and S(GMC)-VOPs

Code	MB type	cbpc (56)
1	0	00
0011	0	01
0010	0	10
0001 01	0	11
011	1	00
0000 111	1	01
0000 110	1	10
0000 0010 1	1	11
010	2	00
0000 101	2	01
0000 100	2	10
0000 0101	2	11
0001 1	3	00
0000 0100	3	01
0000 0011	3	10
0000 011	3	11
0001 00	4	00
0000 0010 0	4	01
0000 0001 1	4	10
0000 0001 0	4	11
0000 0000 1	Stuffing	--

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.8 — VLC table for cbpy in the case of four non-transparent blocks

Code	cbpy(intra-MB) (12 34)	cbpy(inter-MB), (12 34)
0011	00 00	11 11
0010 1	00 01	11 10
0010 0	00 10	11 01
1001	00 11	11 00
0001 1	01 00	10 11
0111	01 01	10 10
0000 10	01 10	10 01
1011	01 11	10 00
0001 0	10 00	01 11
0000 11	10 01	01 10
0101	10 10	01 01
1010	10 11	01 00
0100	11 00	00 11
1000	11 01	00 10
0110	11 10	00 01
11	11 11	00 00

--`,,`,,-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.9 — VLC table for cbpy in the case of three non transparent blocks

Code	cbpy (intra-MB)	cbpy (inter-MB)
011	000	111
000001	001	110
00001	010	101
010	011	100
00010	100	011
00011	101	010
001	110	001
1	111	000

Table B.10 — VLC table for cbpy in the case of two non transparent blocks

Code	cbpy (intra-MB)	cbpy (inter-MB)
0001	00	11
001	01	10
01	10	01
1	11	00

Table B.11 — VLC table for cbpy in the case of one non transparent block

Code	cbpy (intra-MB)	cbpy (inter-MB)
01	0	1
1	1	0

B.1.3 Motion vector

Table B.12 — VLC table for MVD

Codes	Vector differences
0000 0000 0010 1	-16
0000 0000 0011 1	-15.5
0000 0000 0101	-15
0000 0000 0111	-14.5
0000 0000 1001	-14
0000 0000 1011	-13.5
0000 0000 1101	-13
0000 0000 1111	-12.5
0000 0001 001	-12
0000 0001 011	-11.5
0000 0001 101	-11
0000 0001 111	-10.5
0000 0010 001	-10

--`,,`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

0000 0010 011	-9.5
0000 0010 101	-9
0000 0010 111	-8.5
0000 0011 001	-8
0000 0011 011	-7.5
0000 0011 101	-7
0000 0011 111	-6.5
0000 0100 001	-6
0000 0100 011	-5.5
0000 0100 11	-5
0000 0101 01	-4.5
0000 0101 11	-4
0000 0111	-3.5
0000 1001	-3
0000 1011	-2.5
0000 111	-2
0001 1	-1.5
0011	-1
011	-0.5
1	0
010	0.5
0010	1
0001 0	1.5
0000 110	2
0000 1010	2.5
0000 1000	3
0000 0110	3.5
0000 0101 10	4
0000 0101 00	4.5
0000 0100 10	5
0000 0100 010	5.5
0000 0100 000	6
0000 0011 110	6.5
0000 0011 100	7
0000 0011 010	7.5
0000 0011 000	8
0000 0010 110	8.5
0000 0010 100	9
0000 0010 010	9.5
0000 0010 000	10
0000 0001 110	10.5
0000 0001 100	11
0000 0001 010	11.5
0000 0001 000	12
0000 0000 1110	12.5
0000 0000 1100	13
0000 0000 1010	13.5
0000 0000 1000	14

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

0000 0000 0110	14.5
0000 0000 0100	15
0000 0000 0011 0	15.5
0000 0000 0010 0	16

NOTE The last code shall not be used when vop_fcode=1.

B.1.4 DCT coefficients

Table B.13 — Variable length codes for dct_dc_size_luminance

Variable length code	dct_dc_size_luminance
011	0
11	1
10	2
010	3
001	4
0001	5
0000 1	6
0000 01	7
0000 001	8
0000 0001	9
0000 0000 1	10
0000 0000 01	11
0000 0000 001	12

NOTE The variable length code for dct_dc_size_luminance of 10, 11 and 12 are not valid for any object types where the pixel depth is 8 bits. They shall not be present in a bitstream conforming to these object types.

Table B.14 — Variable length codes for dct_dc_size_chrominance

Variable length code	dct_dc_size_chrominance
11	0
10	1
01	2
001	3
0001	4
0000 1	5
0000 01	6
0000 001	7
0000 0001	8
0000 0000 1	9
0000 0000 01	10
0000 0000 001	11
0000 0000 0001	12

NOTE The variable length code for dct_dc_size_chrominance of 10, 11 and 12 are not valid for any object types where the pixel depth is 8 bits. They shall not be present in a bitstream conforming to these object types.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.15 — Differential DC additional codes

Additional code	Differential DC	Size
000000000000 to 011111111111 *	-4095 to -2048	12
00000000000 to 01111111111 *	-2047 to -1024	11
0000000000 to 0111111111 *	-1023 to -512	10
000000000 to 011111111 *	-511 to -256	9
00000000 to 01111111	-255 to -128	8
0000000 to 0111111	-127 to -64	7
000000 to 011111	-63 to -32	6
00000 to 01111	-31 to -16	5
0000 to 0111	-15 to -8	4
000 to 011	-7 to -4	3
00 to 01	-3 to -2	2
0	-1	1
	0	0
1	1	1
10 to 11	2 to 3	2
100 to 111	4 to 7	3
1000 to 1111	8 to 15	4
10000 to 11111	16 to 31	5
100000 to 111111	32 to 63	6
1000000 to 1111111	64 to 127	7
10000000 to 11111111	128 to 255	8
100000000 to 111111111 *	256 to 511	9
1000000000 to 1111111111 *	512 to 1023	10
10000000000 to 11111111111 *	1024 to 2047	11
100000000000 to 111111111111 *	2048 to 4095	12

NOTE 1 The variable length code for “Size” of 10, 11 and 12 are not valid for any object types where the pixel depth is 8 bits. They shall not be present in a bitstream conforming to these object types.

NOTE 2 In cases where dct_dc_size is greater than 8, marked ‘*’ in Table B.15, a marker bit is inserted after the dct_dc_additional_code to prevent start code emulations.

Table B.16 — VLC Table for Intra Luminance and Chrominance Tcoef

VLC CODE	LAST	RUN	LEVEL
10s	0	0	1
1111 s	0	0	3
0101 01s	0	0	6
0010 111s	0	0	9
0001 1111 s	0	0	10
0001 0010 1s	0	0	13
0001 0010 0s	0	0	14
0000 1000 01s	0	0	17
0000 1000 00s	0	0	18
0000 0000 111s	0	0	21
0000 0000 110s	0	0	22
0000 0100 000s	0	0	23
110s	0	0	2
0101 00s	0	1	2

VLC CODE	LAST	RUN	LEVEL
0111 s	1	0	1
0000 1100 1s	0	11	1
0000 0000 101s	1	0	6
0011 11s	1	1	1
0000 0000 100s	1	0	7
0011 10s	1	2	1
0011 01s	0	5	1
0011 00s	1	0	2
0010 011s	1	5	1
0010 010s	0	6	1
0010 001s	1	3	1
0010 000s	1	4	1
0001 1010 s	1	9	1
0001 1001 s	0	8	1

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

VLC CODE	LAST	RUN	LEVEL
0001 1110 s	0	0	11
0000 0011 11s	0	0	19
0000 0100 001s	0	0	24
0000 0101 0000s	0	0	25
1110 s	0	1	1
0001 1101 s	0	0	12
0000 0011 10s	0	0	20
0000 0101 0001s	0	0	26
0110 1s	0	0	4
0001 0001 1s	0	0	15
0000 0011 01s	0	1	7
0110 0s	0	0	5
0001 0001 0s	0	4	2
0000 0101 0010s	0	0	27
0101 1s	0	2	1
0000 0011 00s	0	2	4
0000 0101 0011s	0	1	9
0100 11s	0	0	7
0000 0010 11s	0	3	4
0000 0101 0100s	0	6	3
0100 10s	0	0	8
0000 0010 10s	0	4	3
0100 01s	0	3	1
0000 0010 01s	0	8	2
0100 00s	0	4	1
0000 0010 00s	0	5	3
0010 110s	0	1	3
0000 0101 0101s	0	1	10
0010 101s	0	2	2
0010 100s	0	7	1
0001 1100 s	0	1	4
0001 1011 s	0	3	2
0001 0000 1s	0	0	16
0001 0000 0s	0	1	5
0000 1111 1s	0	1	6
0000 1111 0s	0	2	3
0000 1110 1s	0	3	3
0000 1110 0s	0	5	2
0000 1101 1s	0	6	2
0000 1101 0s	0	7	2
0000 0100 010s	0	1	8
0000 0100 011s	0	9	2
0000 0101 0110s	0	2	5
0000 0101 0111s	0	7	3

VLC CODE	LAST	RUN	LEVEL
0001 1000 s	0	9	1
0001 0111 s	0	10	1
0001 0110 s	1	0	3
0001 0101 s	1	6	1
0001 0100 s	1	7	1
0001 0011 s	1	8	1
0000 1100 0s	0	12	1
0000 1011 1s	1	0	4
0000 1011 0s	1	1	2
0000 1010 1s	1	10	1
0000 1010 0s	1	11	1
0000 1001 1s	1	12	1
0000 1001 0s	1	13	1
0000 1000 1s	1	14	1
0000 0001 11s	0	13	1
0000 0001 10s	1	0	5
0000 0001 01s	1	1	3
0000 0001 00s	1	2	2
0000 0100 100s	1	3	2
0000 0100 101s	1	4	2
0000 0100 110s	1	15	1
0000 0100 111s	1	16	1
0000 0101 1000s	0	14	1
0000 0101 1001s	1	0	8
0000 0101 1010s	1	5	2
0000 0101 1011s	1	6	2
0000 0101 1100s	1	17	1
0000 0101 1101s	1	18	1
0000 0101 1110s	1	19	1
0000 0101 1111s	1	20	1
0000 011	escape		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.17 — VLC table for Inter Lumimance and Chrominance Tcoef

VLC CODE	LAST	RUN	LEVEL
10s	0	0	1
1111 s	0	0	2
0101 01s	0	0	3
0010 111s	0	0	4
0001 1111 s	0	0	5
0001 0010 1s	0	0	6
0001 0010 0s	0	0	7
0000 1000 01s	0	0	8
0000 1000 00s	0	0	9
0000 0000 111s	0	0	10
0000 0000 110s	0	0	11
0000 0100 000s	0	0	12
110s	0	1	1
0101 00s	0	1	2
0001 1110 s	0	1	3
0000 0011 11s	0	1	4
0000 0100 001s	0	1	5
0000 0101 0000s	0	1	6
1110 s	0	2	1
0001 1101 s	0	2	2
0000 0011 10s	0	2	3
0000 0101 0001s	0	2	4
0110 1s	0	3	1
0001 0001 1s	0	3	2
0000 0011 01s	0	3	3
0110 0s	0	4	1
0001 0001 0s	0	4	2
0000 0101 0010s	0	4	3
0101 1s	0	5	1
0000 0011 00s	0	5	2
0000 0101 0011s	0	5	3
0100 11s	0	6	1
0000 0010 11s	0	6	2
0000 0101 0100s	0	6	3
0100 10s	0	7	1
0000 0010 10s	0	7	2
0100 01s	0	8	1
0000 0010 01s	0	8	2
0100 00s	0	9	1
0000 0010 00s	0	9	2
0010 110s	0	10	1
0000 0101 0101s	0	10	2
0010 101s	0	11	1
0010 100s	0	12	1
0001 1100 s	0	13	1
0001 1011 s	0	14	1

VLC CODE	LAST	RUN	LEVEL
0111 s	1	0	1
0000 1100 1s	1	0	2
0000 0000 101s	1	0	3
0011 11s	1	1	1
0000 0000 100s	1	1	2
0011 10s	1	2	1
0011 01s	1	3	1
0011 00s	1	4	1
0010 011s	1	5	1
0010 010s	1	6	1
0010 001s	1	7	1
0010 000s	1	8	1
0001 1010 s	1	9	1
0001 1001 s	1	10	1
0001 1000 s	1	11	1
0001 0111 s	1	12	1
0001 0110 s	1	13	1
0001 0101 s	1	14	1
0001 0100 s	1	15	1
0001 0011 s	1	16	1
0000 1100 0s	1	17	1
0000 1011 1s	1	18	1
0000 1011 0s	1	19	1
0000 1010 1s	1	20	1
0000 1010 0s	1	21	1
0000 1001 1s	1	22	1
0000 1001 0s	1	23	1
0000 1000 1s	1	24	1
0000 0001 11s	1	25	1
0000 0001 10s	1	26	1
0000 0001 01s	1	27	1
0000 0001 00s	1	28	1
0000 0100 100s	1	29	1
0000 0100 101s	1	30	1
0000 0100 110s	1	31	1
0000 0100 111s	1	32	1
0000 0101 1000s	1	33	1
0000 0101 1001s	1	34	1
0000 0101 1010s	1	35	1
0000 0101 1011s	1	36	1
0000 0101 1100s	1	37	1
0000 0101 1101s	1	38	1
0000 0101 1110s	1	39	1
0000 0101 1111s	1	40	1
0000 011	escape		

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

0001 0000 1s	0	15	1
0001 0000 0s	0	16	1
0000 1111 1s	0	17	1
0000 1111 0s	0	18	1
0000 1110 1s	0	19	1
0000 1110 0s	0	20	1
0000 1101 1s	0	21	1
0000 1101 0s	0	22	1
0000 0100 010s	0	23	1
0000 0100 011s	0	24	1
0000 0101 0110s	0	25	1
0000 0101 0111s	0	26	1

Table B.18 — FLC table for RUNS and LEVELS

Code	Run
000 000	0
000 001	1
	63

a) FLC code for RUN

Code	Level
forbidden	-2048
1000 0000 0001	-2047
.	.
1111 1111 1110	-2
1111 1111 1111	-1
forbidden	0
0000 0000 0001	1
0000 0000 0010	2
.	.
0111 1111 1111	2047

b) FLC code for LEVEL

Code	Level
forbidden	-128
1000 0001	-127
.	.
1111 1110	-2
1111 1111	-1
forbidden	0
0000 0001	1
0000 0010	2
.	.
0111 1111	127

c) FLC code for LEVEL when short_video_header is 1

Table B.19 — ESCL(a), LMAX values of intra macroblocks

LAST	RUN	LMAX
0	0	27
0	1	10
0	2	5
0	3	4
0	4-7	3
0	8-9	2
0	10-14	1
0	others	N/A

LAST	RUN	LMAX
1	0	8
1	1	3
1	2-6	2
1	7-20	1
1	others	N/A

-`-`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.20 — ESCL(b), LMAX values of inter macroblocks

LAST	RUN	LMAX
0	0	12
0	1	6
0	2	4
0	3-6	3
0	7-10	2
0	11-26	1
0	others	N/A

LAST	RUN	LMAX
1	0	3
1	1	2
1	2-40	1
1	others	N/A

Table B.21 — ESCR(a), RMAX values of intra macroblocks

LAST	LEVEL	RMAX
0	1	14
0	2	9
0	3	7
0	4	3
0	5	2
0	6-10	1
0	11-27	0
0	others	N/A

LAST	LEVEL	RMAX
1	1	20
1	2	6
1	3	1
1	4-8	0
1	others	N/A

Table B.22 — ESCR(b), RMAX values of inter macroblocks

LAST	LEVEL	RMAX
0	1	26
0	2	10
0	3	6
0	4	2
0	5-6	1
0	7-12	0
0	others	N/A

LAST	LEVEL	RMAX
1	1	40
1	2	1
1	3	0
1	others	N/A

Table B.23 — RVLC table for Tcoef

	intra			inter				
INDEX	LAST	RUN	LEVEL	LAST	RUN	LEVEL	BITS	VLC_CODE
0	0	0	1	0	0	1	4	110s
1	0	0	2	0	1	1	4	111s
2	0	1	1	0	0	2	5	0001s
3	0	0	3	0	2	1	5	1010s
4	1	0	1	1	0	1	5	1011s
5	0	2	1	0	0	3	6	00100s
6	0	3	1	0	3	1	6	00101s
7	0	1	2	0	4	1	6	01000s

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

8	0	0	4	0	5	1	6	01001s
9	1	1	1	1	1	1	6	10010s
10	1	2	1	1	2	1	6	10011s
11	0	4	1	0	1	2	7	001100s
12	0	5	1	0	6	1	7	001101s
13	0	0	5	0	7	1	7	010100s
14	0	0	6	0	8	1	7	010101s
15	1	3	1	1	3	1	7	011000s
16	1	4	1	1	4	1	7	011001s
17	1	5	1	1	5	1	7	100010s
18	1	6	1	1	6	1	7	100011s
19	0	6	1	0	0	4	8	0011100s
20	0	7	1	0	2	2	8	0011101s
21	0	2	2	0	9	1	8	0101100s
22	0	1	3	0	10	1	8	0101101s
23	0	0	7	0	11	1	8	0110100s
24	1	7	1	1	7	1	8	0110101s
25	1	8	1	1	8	1	8	0111000s
26	1	9	1	1	9	1	8	0111001s
27	1	10	1	1	10	1	8	1000010s
28	1	11	1	1	11	1	8	1000011s
29	0	8	1	0	0	5	9	00111100s
30	0	9	1	0	0	6	9	00111101s
31	0	3	2	0	1	3	9	01011100s
32	0	4	2	0	3	2	9	01011101s
33	0	1	4	0	4	2	9	01101100s
34	0	1	5	0	12	1	9	01101101s
35	0	0	8	0	13	1	9	01110100s
36	0	0	9	0	14	1	9	01110101s
37	1	0	2	1	0	2	9	01111000s
38	1	12	1	1	12	1	9	01111001s
39	1	13	1	1	13	1	9	10000010s
40	1	14	1	1	14	1	9	10000011s
41	0	10	1	0	0	7	10	001111100s
42	0	5	2	0	1	4	10	001111101s
43	0	2	3	0	2	3	10	010111100s
44	0	3	3	0	5	2	10	010111101s
45	0	1	6	0	15	1	10	011011100s
46	0	0	10	0	16	1	10	011011101s
47	0	0	11	0	17	1	10	011101100s
48	1	1	2	1	1	2	10	011101101s
49	1	15	1	1	15	1	10	011110100s
50	1	16	1	1	16	1	10	011110101s
51	1	17	1	1	17	1	10	011111000s
52	1	18	1	1	18	1	10	011111001s
53	1	19	1	1	19	1	10	100000010s
54	1	20	1	1	20	1	10	100000011s
55	0	11	1	0	0	8	11	0011111100s

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

56	0	12	1	0	0	9	11	0011111101s
57	0	6	2	0	1	5	11	0101111100s
58	0	7	2	0	3	3	11	0101111101s
59	0	8	2	0	6	2	11	0110111100s
60	0	4	3	0	7	2	11	0110111101s
61	0	2	4	0	8	2	11	0111011100s
62	0	1	7	0	9	2	11	0111011101s
63	0	0	12	0	18	1	11	0111101100s
64	0	0	13	0	19	1	11	0111101101s
65	0	0	14	0	20	1	11	0111110100s
66	1	21	1	1	21	1	11	0111110101s
67	1	22	1	1	22	1	11	0111111000s
68	1	23	1	1	23	1	11	0111111001s
69	1	24	1	1	24	1	11	1000000010s
70	1	25	1	1	25	1	11	1000000011s
71	0	13	1	0	0	10	12	00111111100s
72	0	9	2	0	0	11	12	00111111101s
73	0	5	3	0	1	6	12	01011111100s
74	0	6	3	0	2	4	12	01011111101s
75	0	7	3	0	4	3	12	01101111100s
76	0	3	4	0	5	3	12	01101111101s
77	0	2	5	0	10	2	12	01110111100s
78	0	2	6	0	21	1	12	01110111101s
79	0	1	8	0	22	1	12	01111011100s
80	0	1	9	0	23	1	12	01111011101s
81	0	0	15	0	24	1	12	01111101100s
82	0	0	16	0	25	1	12	01111101101s
83	0	0	17	0	26	1	12	01111110100s
84	1	0	3	1	0	3	12	01111110101s
85	1	2	2	1	2	2	12	01111111000s
86	1	26	1	1	26	1	12	01111111001s
87	1	27	1	1	27	1	12	10000000010s
88	1	28	1	1	28	1	12	10000000011s
89	0	10	2	0	0	12	13	001111111100s
90	0	4	4	0	1	7	13	001111111101s
91	0	5	4	0	2	5	13	010111111100s
92	0	6	4	0	3	4	13	010111111101s
93	0	3	5	0	6	3	13	011011111100s
94	0	4	5	0	7	3	13	011011111101s
95	0	1	10	0	11	2	13	011101111100s
96	0	0	18	0	27	1	13	011101111101s
97	0	0	19	0	28	1	13	011110111100s
98	0	0	22	0	29	1	13	011110111101s
99	1	1	3	1	1	3	13	011111011100s
100	1	3	2	1	3	2	13	011111011101s
101	1	4	2	1	4	2	13	011111101100s
102	1	29	1	1	29	1	13	011111101101s
103	1	30	1	1	30	1	13	011111110100s

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

104	1	31	1	1	31	1	13	011111110101s
105	1	32	1	1	32	1	13	011111111000s
106	1	33	1	1	33	1	13	011111111001s
107	1	34	1	1	34	1	13	100000000010s
108	1	35	1	1	35	1	13	100000000011s
109	0	14	1	0	0	13	14	0011111111100s
110	0	15	1	0	0	14	14	0011111111101s
111	0	11	2	0	0	15	14	0101111111100s
112	0	8	3	0	0	16	14	0101111111101s
113	0	9	3	0	1	8	14	0110111111100s
114	0	7	4	0	3	5	14	0110111111101s
115	0	3	6	0	4	4	14	0111011111100s
116	0	2	7	0	5	4	14	0111011111101s
117	0	2	8	0	8	3	14	0111101111100s
118	0	2	9	0	12	2	14	0111101111101s
119	0	1	11	0	30	1	14	0111110111100s
120	0	0	20	0	31	1	14	0111110111101s
121	0	0	21	0	32	1	14	0111111011100s
122	0	0	23	0	33	1	14	0111111011101s
123	1	0	4	1	0	4	14	0111111101100s
124	1	5	2	1	5	2	14	0111111101101s
125	1	6	2	1	6	2	14	0111111110100s
126	1	7	2	1	7	2	14	0111111110101s
127	1	8	2	1	8	2	14	0111111111000s
128	1	9	2	1	9	2	14	0111111111001s
129	1	36	1	1	36	1	14	1000000000010s
130	1	37	1	1	37	1	14	1000000000011s
131	0	16	1	0	0	17	15	00111111111100s
132	0	17	1	0	0	18	15	00111111111101s
133	0	18	1	0	1	9	15	01011111111100s
134	0	8	4	0	1	10	15	01011111111101s
135	0	5	5	0	2	6	15	01101111111100s
136	0	4	6	0	2	7	15	01101111111101s
137	0	5	6	0	3	6	15	01110111111100s
138	0	3	7	0	6	4	15	01110111111101s
139	0	3	8	0	9	3	15	01111011111100s
140	0	2	10	0	13	2	15	01111011111101s
141	0	2	11	0	14	2	15	01111101111100s
142	0	1	12	0	15	2	15	01111101111101s
143	0	1	13	0	16	2	15	01111110111100s
144	0	0	24	0	34	1	15	01111110111101s
145	0	0	25	0	35	1	15	01111111011100s
146	0	0	26	0	36	1	15	01111111011101s
147	1	0	5	1	0	5	15	01111111101100s
148	1	1	4	1	1	4	15	01111111101101s
149	1	10	2	1	10	2	15	01111111110100s
150	1	11	2	1	11	2	15	01111111110101s
151	1	12	2	1	12	2	15	01111111111000s

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

152	1	38	1	1	38	1	15	01111111111001s
153	1	39	1	1	39	1	15	10000000000010s
154	1	40	1	1	40	1	15	10000000000011s
155	0	0	27	0	0	19	16	001111111111100s
156	0	3	9	0	3	7	16	001111111111101s
157	0	6	5	0	4	5	16	010111111111100s
158	0	7	5	0	7	4	16	010111111111101s
159	0	9	4	0	17	2	16	011011111111100s
160	0	12	2	0	37	1	16	011011111111101s
161	0	19	1	0	38	1	16	011101111111100s
162	1	1	5	1	1	5	16	011101111111101s
163	1	2	3	1	2	3	16	011110111111100s
164	1	13	2	1	13	2	16	011110111111101s
165	1	41	1	1	41	1	16	011111011111100s
166	1	42	1	1	42	1	16	011111011111101s
167	1	43	1	1	43	1	16	011111101111100s
168	1	44	1	1	44	1	16	011111101111101s
169	ESCAPE						5	0000s

ESCAPE code is added at the beginning and the end of these fixed-length codes for realizing two-way decode as shown below. A marker bit is inserted before and after the 11-bit-LEVEL in order to avoid the resync_marker emulation.

ESCAPE	LAST	RUN	marker bit	LEVEL	marker bit	ESCAPE
00001	x	xxxxxx	1	xxxxxxxxxxx	1	0000s

NOTE There are two types for ESCAPE added at the end of these fixed-length codes, and codewords are “0000s”. Also, S=0 : LEVEL is positive and S=1 : LEVEL is negative.

Table B.24 — FLC table for RUN

RUN	CODE
0	000000
1	000001
2	000010
:	:
63	111111

Table B.25 — FLC table for LEVEL

LEVEL	CODE
0	FORBIDDEN
1	00000000001
2	00000000010
:	:
2047	11111111111

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

B.1.5 Shape Coding

Table B.26 — Meaning of shape mode

Index	Shape mode
0	= “MVDs==0 && No Update”
1	= “MVDs!=0 && No Update”
2	transparent
3	opaque
4	“intraCAE”
5	“interCAE && MVDs==0”
6	“interCAE && MVDs!=0”

Table B.27 — bab_type for I-VOP

Index	(2)	(3)	(4)	Index	(2)	(3)	(4)
0	1	001	01	41	001	01	1
1	001	01	1	42	1	01	001
2	01	001	1	43	001	1	01
3	1	001	01	44	001	01	1
4	1	01	001	45	1	01	001
5	1	01	001	46	001	01	1
6	1	001	01	47	01	001	1
7	1	01	001	48	1	01	001
8	01	001	1	49	001	01	1
9	001	01	1	50	01	001	1
10	1	01	001	51	1	001	01
11	1	01	001	52	001	1	01
12	001	01	1	53	01	001	1
13	1	01	001	54	1	001	01
14	01	1	001	55	01	001	1
15	001	01	1	56	01	001	1
16	1	01	001	57	1	01	001
17	1	01	001	58	1	01	001
18	01	001	1	59	1	01	001
19	1	01	001	60	1	01	001
20	001	01	1	61	1	01	001
21	01	001	1	62	01	001	1
22	1	01	001	63	1	01	001
23	001	01	1	64	001	01	1
24	01	001	1	65	001	01	1
25	001	01	1	66	01	001	1
26	001	01	1	67	001	1	01
27	1	01	001	68	001	1	01
28	1	01	001	69	01	001	1
29	1	01	001	70	001	1	01
30	1	01	001	71	001	01	1
31	1	01	001	72	1	001	01

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

32	1	01	001	73	001	01	1
33	1	01	001	74	01	001	1
34	1	01	001	75	01	001	1
35	001	01	1	76	001	1	01
36	1	01	001	77	001	01	1
37	001	01	1	78	1	001	01
38	001	01	1	79	001	1	01
39	1	01	001	80	001	01	1
40	001	1	01				

Table B.28 — bab_type for P-VOP, B-VOP, and S(GMC)-VOP

		bab_type in current VOP						
		0	**1**	**2**	**3**	**4**	**5**	**6**
bab_type in reference VOP	**0**	1	01	00001	000001	0001	0010	0011
	1	01	1	00001	000001	001	0000001	0001
	2	0001	001	1	000001	01	0000001	00001
	3	1	0001	000001	001	01	0000001	00001
	4	011	001	0001	00001	1	000001	010
	5	01	0001	00001	000001	001	11	10
	6	001	0001	00001	000001	01	10	11

Table B.29 — VLC table for MVDs

MVDs	**Codes**
0	0
±1	10s
±2	110s
±3	1110s
±4	11110s
±5	111110s
±6	1111110s
±7	11111110s
±8	111111110s
±9	1111111110s
±10	11111111110s
±11	111111111110s
±12	1111111111110s
±13	11111111111110s
±14	111111111111110s
±15	1111111111111110s
±16	11111111111111110s

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.30 — VLC table for MVDs (Horizontal element is 0)

MVDs	Codes
±1	0s
±2	10s
±3	110s
±4	1110s
±5	11110s
±6	111110s
±7	1111110s
±8	11111110s
±9	111111110s
±10	1111111110s
±11	11111111110s
±12	111111111110s
±13	1111111111110s
±14	11111111111110s
±15	111111111111110s
±16	1111111111111110s
s: sign bit (if MVDs is positive s="1", otherwise s="0").	

Table B.31 — VLC for conv_ratio

conv_ratio	Code
1	0
2	10
4	11

These tables contain the probabilities for a binary alpha pixel being equal to 0 for intra and inter shape coding using CAE. All probabilities are normalised to the range [1,65535].

As an example, given an INTRA context number C, the probability that the pixel is zero is given by intra_prob[C].

Table B.32 — Probabilities for arithmetic decoding of shape

USInt intra_prob[1024] = {

65267,16468,65003,17912,64573,8556,64252,5653,40174,3932,29789,277,45152,1140,32768,2043,
4499,80,6554,1144,21065,465,32768,799,5482,183,7282,264,5336,99,6554,563,
54784,30201,58254,9879,54613,3069,32768,58495,32768,32768,32768,2849,58982,54613,32768,12892,
31006,1332,49152,3287,60075,350,32768,712,39322,760,32768,354,52659,432,61854,150,
64999,28362,65323,42521,63572,32768,63677,18319,4910,32768,64238,434,53248,32768,61865,13590,
16384,32768,13107,333,32768,32768,32768,32768,32768,32768,1074,780,25058,5461,6697,233,
62949,30247,63702,24638,59578,32768,32768,42257,32768,32768,49152,546,62557,32768,54613,19258,
62405,32569,64600,865,60495,10923,32768,898,34193,24576,64111,341,47492,5231,55474,591,
65114,60075,64080,5334,65448,61882,64543,13209,54906,16384,35289,4933,48645,9614,55351,7318,
49807,54613,32768,32768,50972,32768,32768,32768,15159,1928,2048,171,3093,8,6096,74,
32768,60855,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,55454,32768,57672,
32768,16384,32768,21845,32768,32768,32768,32768,32768,32768,32768,5041,28440,91,32768,45,

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

65124,10923,64874,5041,65429,57344,63435,48060,61440,32768,63488,24887,59688,3277,63918,14021,
32768,32768,32768,32768,32768,32768,32768,32768,690,32768,32768,1456,32768,32768,8192,728,
32768,32768,58982,17944,65237,54613,32768,2242,32768,32768,32768,42130,49152,57344,58254,16740,
32768,10923,54613,182,32768,32768,32768,7282,49152,32768,32768,5041,63295,1394,55188,77,
63672,6554,54613,49152,64558,32768,32768,5461,64142,32768,32768,32768,62415,32768,32768,16384,
1481,438,19661,840,33654,3121,64425,6554,4178,2048,32768,2260,5226,1680,32768,565,
60075,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,32768,
16384,261,32768,412,16384,636,32768,4369,23406,4328,32768,524,15604,560,32768,676,
49152,32768,49152,32768,32768,32768,64572,32768,32768,32768,54613,32768,32768,32768,32768,32768,
4681,32768,5617,851,32768,32768,59578,32768,32768,32768,3121,3121,49152,32768,6554,10923,
32768,32768,54613,14043,32768,32768,32768,3449,32768,32768,32768,32768,32768,32768,32768,32768,
57344,32768,57344,3449,32768,32768,32768,3855,58982,10923,32768,239,62259,32768,49152,85,
58778,23831,62888,20922,64311,8192,60075,575,59714,32768,57344,40960,62107,4096,61943,3921,
39862,15338,32768,1524,45123,5958,32768,58982,6669,930,1170,1043,7385,44,8813,5011,
59578,29789,54613,32768,32768,32768,32768,32768,32768,32768,32768,32768,58254,56174,32768,32768,
64080,25891,49152,22528,32768,2731,32768,10923,10923,3283,32768,1748,17827,77,32768,108,
62805,32768,62013,42612,32768,32768,61681,16384,58982,60075,62313,58982,65279,58982,62694,62174,
32768,32768,10923,950,32768,32768,32768,32768,5958,32768,38551,1092,11012,39322,13705,2072,
54613,32768,32768,11398,32768,32768,32768,145,32768,32768,32768,29789,60855,32768,61681,54792,
32768,32768,32768,17348,32768,32768,32768,8192,57344,16384,32768,3582,52581,580,24030,303,
62673,37266,65374,6197,62017,32768,49152,299,54613,32768,32768,32768,35234,119,32768,3855,
31949,32768,32768,49152,16384,32768,32768,32768,24576,32768,49152,32768,17476,32768,32768,57445,
51200,50864,54613,27949,60075,20480,32768,57344,32768,32768,32768,32768,32768,45875,32768,32768,
11498,3244,24576,482,16384,1150,32768,16384,7992,215,32768,1150,23593,927,32768,993,
65353,32768,65465,46741,41870,32768,64596,59578,62087,32768,12619,23406,11833,32768,47720,17476,
32768,32768,2621,6554,32768,32768,32768,32768,32768,32768,5041,32768,16384,32768,4096,2731,
63212,43526,65442,47124,65410,35747,60304,55858,60855,58982,60075,19859,35747,63015,64470,25432,
58689,1118,64717,1339,24576,32768,32768,1257,53297,1928,32768,33,52067,3511,62861,453,
64613,32768,32768,32768,64558,32768,32768,2731,49152,32768,32768,32768,61534,32768,32768,35747,
32768,32768,32768,32768,13107,32768,32768,32768,32768,32768,32768,32768,20480,32768,32768,32768,
32768,32768,32768,54613,40960,5041,32768,32768,32768,32768,32768,3277,64263,57592,32768,3121,
32768,32768,32768,32768,32768,10923,32768,32768,32768,8192,32768,32768,5461,6899,32768,1725,
63351,3855,63608,29127,62415,7282,64626,60855,32768,32768,60075,5958,44961,32768,61866,53718,
32768,32768,32768,32768,32768,32768,6554,32768,32768,32768,32768,32768,2521,978,32768,1489,
58254,32768,58982,61745,21845,32768,54613,58655,60075,32768,49152,16274,50412,64344,61643,43987,
32768,32768,32768,1638,32768,32768,32768,24966,54613,32768,32768,2427,46951,32768,17970,654,
65385,27307,60075,26472,64479,32768,32768,4681,61895,32768,32768,16384,58254,32768,32768,6554,
37630,3277,54613,6554,4965,5958,4681,32768,42765,16384,32768,21845,22827,16384,32768,6554,
65297,64769,60855,12743,63195,16384,32768,37942,32768,32768,32768,32768,60075,32768,62087,54613,
41764,2161,21845,1836,17284,5424,10923,1680,11019,555,32768,431,39819,907,32768,171,
65480,32768,64435,33803,2595,32768,57041,32768,61167,32768,32768,32768,32768,32768,32768,1796,
60855,32768,17246,978,32768,32768,8192,32768,32768,32768,14043,2849,32768,2979,6554,6554,
65507,62415,65384,61891,65273,58982,65461,55097,32768,32768,32768,55606,32768,2979,3745,16913,
61885,13827,60893,12196,60855,53248,51493,11243,56656,783,55563,143,63432,7106,52429,445,
65485,1031,65020,1380,65180,57344,65162,36536,61154,6554,26569,2341,63593,3449,65102,533,
47827,2913,57344,3449,35688,1337,32768,22938,25012,910,7944,1008,29319,607,64466,4202,
64549,57301,49152,20025,63351,61167,32768,45542,58982,14564,32768,9362,61895,44840,32768,26385,
59664,17135,60855,13291,40050,12252,32768,7816,25798,1850,60495,2662,18707,122,52538,231,
65332,32768,65210,21693,65113,6554,65141,39667,62259,32768,22258,1337,63636,32768,64255,52429,
60362,32768,6780,819,16384,32768,16384,4681,49152,32768,8985,2521,24410,683,21535,16585,
65416,46091,65292,58328,64626,32768,65016,39897,62687,47332,62805,28948,64284,53620,52870,49567,

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

```
65032,31174,63022,28312,64299,46811,48009,31453,61207,7077,50299,1514,60047,2634,46488,235
};

USInt inter_prob[512] = {
65532,62970,65148,54613,62470,8192,62577,8937,65480,64335,65195,53248,65322,62518,62891,38312,
65075,53405,63980,58982,32768,32768,54613,32768,65238,60009,60075,32768,59294,19661,61203,13107,
63000,9830,62566,58982,11565,32768,25215,3277,53620,50972,63109,43691,54613,32768,39671,17129,
59788,6068,43336,27913,6554,32768,12178,1771,56174,49152,60075,43691,58254,16384,49152,9930,
23130,7282,40960,32768,10923,32768,32768,32768,27307,32768,32768,32768,32768,32768,32768,32768,
36285,12511,10923,32768,45875,16384,32768,32768,16384,23831,4369,32768,8192,10923,32768,32768,
10175,2979,18978,10923,54613,32768,6242,6554,1820,10923,32768,32768,32768,32768,32768,5461,
28459,593,11886,2030,3121,4681,1292,112,42130,23831,49152,29127,32768,6554,5461,2048,
65331,64600,63811,63314,42130,19661,49152,32768,65417,64609,62415,64617,64276,44256,61068,36713,
64887,57525,53620,61375,32768,8192,57344,6554,63608,49809,49152,62623,32768,15851,58982,34162,
55454,51739,64406,64047,32768,32768,7282,32768,49152,58756,62805,64990,32768,14895,16384,19418,
57929,24966,58689,31832,32768,16384,10923,6554,54613,42882,57344,64238,58982,10082,20165,20339,
62687,15061,32768,10923,32768,10923,32768,16384,59578,34427,32768,16384,32768,7825,32768,7282,
58052,23400,32768,5041,32768,2849,32768,32768,47663,15073,57344,4096,32768,1176,32768,1320,
24858,410,24576,923,32768,16384,16384,5461,16384,1365,32768,5461,32768,5699,8192,13107,
46884,2361,23559,424,19661,712,655,182,58637,2094,49152,9362,8192,85,32768,1228,
65486,49152,65186,49152,61320,32768,57088,25206,65352,63047,62623,49152,64641,62165,58986,18304,
64171,16384,60855,54613,42130,32768,61335,32768,58254,58982,49152,32768,60985,35289,64520,31554,
51067,32768,64074,32768,40330,32768,34526,4096,60855,32768,63109,58254,57672,16384,31009,2567,
23406,32768,44620,10923,32768,32768,32099,10923,49152,49152,54613,60075,63422,54613,46388,39719,
58982,32768,54613,32768,14247,32768,22938,5041,32768,49152,32768,32768,25321,6144,29127,10999,
41263,32768,46811,32768,267,4096,426,16384,32768,19275,49152,32768,1008,1437,5767,11275,
5595,5461,37493,6554,4681,32768,6147,1560,38229,10923,32768,40960,35747,2521,5999,312,
17052,2521,18808,3641,213,2427,574,32,51493,42130,42130,53053,11155,312,2069,106,
64406,45197,58982,32768,32768,16384,40960,36864,65336,64244,60075,61681,65269,50748,60340,20515,
58982,23406,57344,32768,6554,16384,19661,61564,60855,47480,32768,54613,46811,21701,54909,37826,
32768,58982,60855,60855,32768,32768,39322,49152,57344,45875,60855,55706,32768,24576,62313,25038,
54613,8192,49152,10923,32768,32768,32768,32768,32768,19661,16384,51493,32768,14043,40050,44651,
59578,5174,32768,6554,32768,5461,23593,5461,63608,51825,32768,23831,58887,24032,57170,3298,
39322,12971,16384,49152,1872,618,13107,2114,58982,25705,32768,60075,28913,949,18312,1815,
48188,114,51493,1542,5461,3855,11360,1163,58982,7215,54613,21487,49152,4590,48430,1421,
28944,1319,6868,324,1456,232,820,7,61681,1864,60855,9922,4369,315,6589,14
};
```

Table B.33 — Probability for arithmetic decoding of spatial scalable binary shape

```
USInt SI_bab_type_prob[1]={ 59808 }


USInt enh_intra_v_prob[128]={
65476, 64428, 62211, 63560, 52253, 58271, 38098, 31981, 50087, 41042,
54620, 31532, 8382, 10754, 3844, 6917, 63834, 50444, 50140, 63043,
58093, 45146, 36768, 13351, 17594, 28777, 39830, 38719, 9768, 21447,
12340,  9786, 60461, 41489, 27433, 53893, 47246, 11415, 13754, 24965,
51620, 28011, 11973, 29709, 13878, 22794, 24385, 1558, 57065, 41918,
25259, 55117, 48064, 12960, 19929, 5937, 25730, 22366, 5204, 32865,
3415, 14814, 6634, 1155, 64444, 62907, 56337, 63144, 38112, 56527,
40247, 37088, 60326, 45675, 51248, 15151, 18868, 43723, 14757, 11721,
```

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

62436, 50971, 51738, 59767, 49927, 50675, 38182, 24724, 48447, 47316,
56628, 36336, 12264, 25893, 24243, 5358, 58717, 56646, 48302, 60515,
36497, 26959, 43579, 40280, 54092, 20741, 10891, 7504, 8109, 30840,
6772, 4090, 59810, 61410, 53216, 64127, 32344, 12462, 23132, 19270,
32232, 24774, 9615, 17750, 1714, 6539, 3237, 152
}

USInt enh_intra_h_prob[128]={
65510, 63321, 63851, 62223, 64959, 62202, 63637, 48019, 57072, 33553,
37041, 9527, 53190, 50479, 54232, 12855, 62779, 63980, 49604, 31847,
57591, 64385, 40657, 8402, 33878, 54743, 17873, 8707, 34470, 54322,
16702, 2192, 58325, 48447, 7345, 31317, 45687, 44236, 16685, 24144,
34327, 18724, 10591, 24965, 9247, 7281, 3144, 5921, 59349, 33539,
11447, 5543, 58082, 48995, 35630, 10653, 7123, 15893, 23830, 800,
3491, 15792, 8930, 905, 65209, 63939, 52634, 62194, 64937, 53948,
60081, 46851, 56157, 50930, 35498, 24655, 56331, 59318, 32209, 6872,
59172, 64273, 46724, 41200, 53619, 59022, 37941, 20529, 55026, 52858,
26402, 45073, 57740, 55485, 20533, 6288, 64286, 55438, 16454, 55656,
61175, 45874, 28536, 53762, 58056, 21895, 5482, 39352, 32635, 21633,
2137, 4016, 58490, 14100, 18724, 10461, 53459, 15490, 57992, 15128,
12034, 4340, 6761, 1859, 5794, 6785, 2412, 35
}

B.1.6 Sprite Coding

Table B.34 — Code table for the first trajectory point

dmv value	SSS	VLC	dmv_code
-16383 … -8192, 8192 … 16383	14	111111111110	00000000000000...01111111111111, 10000000000000...11111111111111
-8191 … -4096, 4096 … 8191	13	11111111110	0000000000000...0111111111111, 1000000000000...1111111111111
-4095 … -2048, 2048 … 4095	12	1111111110	000000000000...011111111111, 100000000000...111111111111
-2047...-1024, 1024...2047	11	111111110	00000000000...01111111111, 10000000000...11111111111
-1023...-512, 512...1023	10	11111110	0000000000...0111111111, 1000000000...1111111111
-511...-256, 256...511	9	1111110	000000000...011111111, 100000000...111111111
-255...-128, 128...255	8	111110	00000000...01111111, 10000000...11111111
-127...-64, 64...127	7	11110	0000000...0111111, 1000000...1111111
-63...-32, 32...63	6	1110	000000...011111, 100000...111111
-31...-16, 16...31	5	110	00000...01111, 10000...1111
-15...-8, 8...15	4	101	0000...0111, 1000...1111
-7...-4, 4...7	3	100	000...011, 100...111
-3...-2, 2...3	2	011	00...01, 10...11
-1, 1	1	010	0, 1
0	0	00	-

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.35 — Code table for scaled brightness change factor

brightness_change_ factor value	brightness_ change_factor_ length value	brightness_ change_factor_ length VLC	brightness_change_factor
-16...-1, 1...16	1	0	00000...01111, 10000...11111
-48...-17, 17...48	2	10	000000...011111, 100000...111111
112...-49, 49...112	3	110	0000000...0111111, 1000000...1111111
113…624	4	1110	000000000...111111111
625...1648	4	1111	0000000000…1111111111

B.1.7 DCT based facial object decoding

Table B.36 — Viseme_select_table, 29 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	6	001000	10	6	010001	20	6	010000
1	6	001001	11	6	011001	21	6	010010
2	6	001011	12	5	00001	22	6	011010
3	6	001101	13	6	011101	23	5	00010
4	6	001111	14	1	1	24	6	011110
5	6	010111	15	6	010101	25	6	010110
6	6	011111	16	6	010100	26	6	001110
7	5	00011	17	6	011100	27	6	001100
8	6	011011	18	5	00000	28	6	001010
9	6	010011	19	6	011000			

Table B.37 — Expression_select_table, 13 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	5	01000	5	4	0011	10	5	01110
1	5	01001	6	1	1	11	5	01100
2	5	01011	7	4	0001	12	5	01010
3	5	01101	8	4	0000			
4	5	01111	9	4	0010			

Table B.38 — Viseme and Expression intensity_table, 127 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	17	10010001101010010	43	16	1001000110100111	86	16	1001000110100110
1	17	10010001101010011	44	8	10011100	87	16	1001000110100100
2	17	10010001101010101	45	11	10010001111	88	16	1001000110100010
3	17	10010001101010111	46	9	100100010	89	16	1001000110100000
4	17	10010001101011001	47	10	1110001011	90	16	1001000110011110
5	17	10010001101011011	48	9	100011011	91	16	1001000110011100
6	17	10010001101011101	49	10	1110001001	92	16	1001000110011010
7	17	10010001101011111	50	9	100011010	93	16	1001000110011000

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8	17	10010001101100001	51	9	100111010	94	16	1001000110010110
9	17	10010001101100011	52	10	1110001000	95	16	1001000110010100
10	17	10010001101100101	53	7	1000111	96	16	1001000110010010
11	17	10010001101100111	54	7	1000010	97	16	1001000110010000
12	17	10010001101101001	55	8	10010000	98	16	1001000110001110
13	17	10010001101101011	56	7	1001111	99	16	1001000110001100
14	17	10010001101101101	57	7	1110000	100	16	1001000110001010
15	17	10010001101101111	58	6	100000	101	16	1001000110001000
16	17	10010001101110001	59	6	100101	102	16	1001000110000110
17	17	10010001101110011	60	6	111010	103	16	1001000110000100
18	17	10010001101110111	61	5	11111	104	16	1001000110000010
19	17	10010001101111001	62	3	101	105	16	1001000110000000
20	17	10010001101111011	63	1	0	106	17	10010001101111110
21	17	10010001101111101	64	3	110	107	17	10010001101111100
22	17	10010001101111111	65	5	11110	108	17	10010001101111010
23	16	1001000110000001	66	6	111001	109	17	10010001101111000
24	16	1001000110000011	67	6	111011	110	17	10010001101110110
25	16	1001000110000101	68	6	100010	111	17	10010001101110010
26	16	1001000110000111	69	7	1001100	112	17	10010001101110000
27	16	1001000110001001	70	7	1001001	113	17	10010001101101110
28	16	1001000110001011	71	7	1001101	114	17	10010001101101100
29	16	1001000110001101	72	8	10001100	115	17	10010001101101010
30	16	1001000110001111	73	8	10000111	116	17	10010001101101000
31	16	1001000110010001	74	8	10000110	117	17	10010001101100110
32	16	1001000110010011	75	17	10010001101110100	118	17	10010001101100100
33	16	1001000110010101	76	9	111000110	119	17	10010001101100010
34	16	1001000110010111	77	11	11100010100	120	17	10010001101100000
35	16	1001000110011001	78	11	10011101111	121	17	10010001101011110
36	16	1001000110011011	79	17	10010001101110101	122	17	10010001101011100
37	16	1001000110011101	80	10	1001110110	123	17	10010001101011010
38	16	1001000110011111	81	16	1001000110101000	124	17	10010001101011000
39	16	1001000110100001	82	11	10010001110	125	17	10010001101010110
40	16	1001000110100011	83	10	1110001111	126	17	10010001101010100
41	11	11100010101	84	11	10011101110			
42	16	1001000110100101	85	10	1110001110			

Table B.39 — Runlength_table, 16 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	1	1	6	9	000001011	12	8	00000000
1	2	01	7	9	000001101	13	8	00000010
2	3	001	8	9	000001111	14	9	000001110
3	4	0001	9	8	00000011	15	9	000001100
4	5	00001	10	8	00000001			
5	9	000001010	11	8	00000100			

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.40 — DC_table, 512 symbols

symbol	bits	code	symbol	bits	code	symbol	bits	code
0	17	11010111001101010	171	17	11010111001111001	342	17	11010111001111000
1	17	11010111001101011	172	17	11010111010000001	343	17	11010111001110000
2	17	11010111001101101	173	17	11010111010001001	344	17	11010111001110010
3	17	11010111001101111	174	17	11010111010010001	345	17	11010111001111010
4	17	11010111001110101	175	17	11010111010011001	346	17	11010111010000010
5	17	11010111001110111	176	17	11010111010101001	347	17	11010111010001010
6	17	11010111001111101	177	17	11010111010110001	348	17	11010111010010010
7	17	11010111001111111	178	17	11010111010111001	349	17	11010111010011010
8	17	11010111010000101	179	17	11010111011000001	350	17	11010111010101010
9	17	11010111010000111	180	17	11010111011001001	351	17	11010111010110010
10	17	11010111010001101	181	17	11010111011011001	352	17	11010111010111010
11	17	11010111010001111	182	17	11010111011111001	353	17	11010111011000010
12	17	11010111010010101	183	17	11010111100000001	354	17	11010111011001010
13	17	11010111010010111	184	17	11010111100001001	355	17	11010111011011010
14	17	11010111010011101	185	17	11010111100011001	356	17	11010111011111010
15	17	11010111010011111	186	17	11010111100100001	357	17	11010111100000010
16	17	11010111010101101	187	17	11010111100101001	358	17	11010111100001010
17	17	11010111010101111	188	17	11010111100111001	359	17	11010111100011010
18	17	11010111010110111	189	17	11010111101000001	360	17	11010111100100010
19	17	11010111010111101	190	17	11010111101001001	361	17	11010111100101010
20	17	11010111010111111	191	17	11010111101011001	362	17	11010111100111010
21	17	11010111011000111	192	17	11010111101111001	363	17	11010111101000010
22	17	11010111011001101	193	17	11010111110000001	364	17	11010111101001010
23	17	11010111011001111	194	17	11010111110001001	365	17	11010111101011010
24	17	11010111011011101	195	17	11010111110011001	366	17	11010111101111010
25	17	11010111011011111	196	17	11010111110111001	367	17	11010111110000010
26	17	11010111011111101	197	17	11010111111100001	368	17	11010111110001010
27	17	11010111011111111	198	17	11010111111101001	369	17	11010111110011010
28	17	11010111100000111	199	17	11010111111111001	370	17	11010111110111010
29	17	11010111100001101	200	16	1101011100000001	371	17	11010111111100010
30	17	11010111100001111	201	16	1101011100001001	372	17	11010111111101010
31	17	11010111100011101	202	16	1101011100011001	373	17	11010111111111010
32	17	11010111100011111	203	17	11010111111001001	374	16	1101011100000010
33	17	11010111100100101	204	17	11010111111010001	375	16	1101011100001010
34	17	11010111100100111	205	17	11010111111011001	376	16	1101011100011010
35	17	11010111100101101	206	16	1101011100101001	377	17	11010111111001010
36	17	11010111100101111	207	17	11010111110100001	378	17	11010111111010010
37	17	11010111100111101	208	17	11010111110101001	379	17	11010111111011010
38	17	11010111100111111	209	17	11010111101101001	380	16	1101011100101010
39	17	11010111101000101	210	17	11010111011100001	381	17	11010111110100010
40	17	11010111101000111	211	16	1101011100100000	382	17	11010111110101010
41	17	11010111101001101	212	16	1101011100100001	383	17	11010111101101010
42	17	11010111101001111	213	17	11010111111000001	384	17	11010111011100010
43	17	11010111101011101	214	16	1101011100010001	385	17	11010111011101010
44	17	11010111101011111	215	17	11010111111110001	386	17	11010111011101000

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

45	17	11010111101111101	216	17	11010111110110001	387	16	1101011100100010
46	17	11010111101111111	217	17	11010111110010001	388	17	11010111111000010
47	17	11010111110000101	218	11	11101100101	389	16	1101011100010010
48	17	11010111110000111	219	11	11011111011	390	17	11010111111110010
49	17	11010111110001101	220	11	11011110001	391	17	11010111110110010
50	17	11010111110001111	221	10	1101110011	392	17	11010111110010010
51	17	11010111110011101	222	17	11010111101110001	393	17	11010111101110010
52	17	11010111110011111	223	17	11010111010100000	394	17	11010111101010010
53	17	11010111110111101	224	17	11010111010100001	395	17	11010111101010000
54	17	11010111110111111	225	17	11010111011110100	396	17	11010111100010010
55	17	11010111111100101	226	17	11010111011110101	397	17	11010111100010000
56	17	11010111111100111	227	17	11010111011110001	398	17	11010111011010010
57	17	11010111111101101	228	17	11010111100010101	399	17	11010111011010000
58	17	11010111111101111	229	17	11010111100110000	400	16	1101011100110010
59	17	11010111111111101	230	17	11010111100110001	401	16	1101011100110000
60	17	11010111111111111	231	17	11010111101010101	402	17	11010111010100110
61	16	1101011100000101	232	11	11101100111	403	17	11010111010100100
62	16	1101011100000111	233	17	11010111101110101	404	17	11010111010100010
63	16	1101011100001101	234	11	11101100110	405	17	11010111011010110
64	16	1101011100001111	235	17	11010111110110101	406	17	11010111011010100
65	16	1101011100011101	236	17	11010111111000100	407	17	11010111011110110
66	16	1101011100011111	237	8	11010110	408	17	11010111011110010
67	17	11010111111001101	238	11	11011110010	409	17	11010111100010110
68	17	11010111111001111	239	9	110010100	410	17	11010111100110110
69	17	11010111111010101	240	10	1101110001	411	17	11010111100110100
70	17	11010111111010111	241	9	110001111	412	17	11010111100110010
71	17	11010111111011101	242	10	1101111100	413	17	11010111101010110
72	17	11010111111011111	243	9	110010101	414	17	11010111101110110
73	16	1101011100101101	244	9	110111111	415	17	11010111110010110
74	16	1101011100101111	245	10	1101110100	416	17	11010111110010100
75	17	11010111110100101	246	7	1100100	417	17	11010111110110110
76	17	11010111110100111	247	8	11101101	418	17	11010111111110110
77	17	11010111110101101	248	8	11001011	419	17	11010111111110100
78	17	11010111110101111	249	7	1101100	420	16	1101011100010110
79	17	11010111101101101	250	7	1101101	421	16	1101011100010100
80	17	11010111101101111	251	7	1110111	422	17	11010111111000110
81	17	11010111011100101	252	6	110100	423	16	1101011100100110
82	17	11010111011100111	253	6	111001	424	16	1101011100100100
83	17	11010111011101101	254	5	11111	425	17	11010111101100110
84	17	11010111011101111	255	3	100	426	17	11010111101100100
85	17	11010111101100001	256	1	0	427	17	11010111101100010
86	17	11010111101100011	257	3	101	428	17	11010111101100000
87	17	11010111101100101	258	5	11110	429	17	11010111011101110
88	17	11010111101100111	259	6	111000	430	17	11010111011101100
89	16	1101011100100101	260	6	111010	431	17	11010111011100110
90	16	1101011100100111	261	6	110000	432	17	11010111011100100
91	17	11010111111000111	262	7	1100111	433	17	11010111101101110
92	16	1101011100010101	263	7	1100110	434	17	11010111101101100

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

93	16	1101011100010111	264	7	1101010	435	17	11010111110101110
94	17	11010111111110101	265	8	11000101	436	17	11010111110101100
95	17	11010111111110111	266	8	11000110	437	17	11010111110100110
96	17	11010111110110111	267	8	11000100	438	17	11010111110100100
97	17	11010111110010101	268	17	11010111111000101	439	16	1101011100101110
98	17	11010111110010111	269	9	111011000	440	16	1101011100101100
99	17	11010111101110111	270	11	11011111010	441	17	11010111111011110
100	17	11010111101010111	271	11	11011110101	442	17	11010111111011100
101	17	11010111100110011	272	17	11010111100000101	443	17	11010111111010110
102	17	11010111100110101	273	10	1101111011	444	17	11010111111010100
103	17	11010111100110111	274	17	11010111011000101	445	17	11010111111001110
104	17	11010111100010111	275	11	11011110011	446	17	11010111111001100
105	17	11010111011110011	276	9	110001110	447	16	1101011100011110
106	17	11010111011110111	277	11	11011110000	448	16	1101011100011100
107	17	11010111011010101	278	10	1101110111	449	16	1101011100001110
108	17	11010111011010111	279	17	11010111010110101	450	16	1101011100001100
109	17	11010111010100011	280	16	1101011100110100	451	16	1101011100000110
110	17	11010111010100101	281	10	1101110010	452	16	1101011100000100
111	17	11010111010100111	282	10	1101110000	453	17	11010111111111110
112	16	1101011100110001	283	11	11011101010	454	17	11010111111111100
113	16	1101011100110011	284	17	11010111010110100	455	17	11010111111101110
114	17	11010111011010001	285	17	11010111011000100	456	17	11010111111101100
115	17	11010111011010011	286	17	11010111100000100	457	17	11010111111100110
116	17	11010111100010001	287	11	11011101100	458	17	11010111111100100
117	17	11010111100010011	288	17	11010111110110100	459	17	11010111110111110
118	17	11010111101010001	289	17	11010111101110100	460	17	11010111110111100
119	17	11010111101010011	290	17	11010111101010100	461	17	11010111110011110
120	17	11010111101110011	291	11	11101100100	462	17	11010111110011100
121	17	11010111110010011	292	17	11010111100010100	463	17	11010111110001110
122	17	11010111110110011	293	17	11010111011110000	464	17	11010111110001100
123	17	11010111111110011	294	11	11011110100	465	17	11010111110000110
124	16	1101011100010011	295	11	11011101011	466	17	11010111110000100
125	17	11010111111000011	296	17	11010111101110000	467	17	11010111101111110
126	16	1101011100100011	297	17	11010111110010000	468	17	11010111101111100
127	17	11010111011101001	298	17	11010111110110000	469	17	11010111101011110
128	17	11010111011101011	299	17	11010111111110000	470	17	11010111101011100
129	17	11010111011100011	300	16	1101011100010000	471	17	11010111101001110
130	17	11010111101101011	301	17	11010111111000000	472	17	11010111101001100
131	17	11010111110101011	302	11	11011101101	473	17	11010111101000110
132	17	11010111110100011	303	17	11010111011100000	474	17	11010111101000100
133	16	1101011100101011	304	17	11010111101101000	475	17	11010111100111110
134	17	11010111111011011	305	17	11010111110101000	476	17	11010111100111100
135	17	11010111111010011	306	17	11010111110100000	477	17	11010111100101110
136	17	11010111111001011	307	16	1101011100101000	478	17	11010111100101100
137	16	1101011100011011	308	17	11010111111011000	479	17	11010111100100110
138	16	1101011100001011	309	17	11010111111010000	480	17	11010111100100100
139	16	1101011100000011	310	17	11010111111001000	481	17	11010111100011110
140	17	11010111111111011	311	16	1101011100011000	482	17	11010111100011100

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

141	17	11010111111101011	312	16	1101011100001000	483	17	11010111100001110
142	17	11010111111100011	313	16	1101011100000000	484	17	11010111100001100
143	17	11010111110111011	314	17	11010111111111000	485	17	11010111100000110
144	17	11010111110011011	315	17	11010111111101000	486	17	11010111011111110
145	17	11010111110001011	316	17	11010111111100000	487	17	11010111011111100
146	17	11010111110000011	317	17	11010111110111000	488	17	11010111011011110
147	17	11010111101111011	318	17	11010111110011000	489	17	11010111011011100
148	17	11010111101011011	319	17	11010111110001000	490	17	11010111011001110
149	17	11010111101001011	320	17	11010111110000000	491	17	11010111011001100
150	17	11010111101000011	321	17	11010111101111000	492	17	11010111011000110
151	17	11010111100111011	322	17	11010111101011000	493	17	11010111010111110
152	17	11010111100101011	323	17	11010111101001000	494	17	11010111010111100
153	17	11010111100100011	324	17	11010111101000000	495	17	11010111010110110
154	17	11010111100011011	325	17	11010111100111000	496	17	11010111010101110
155	17	11010111100001011	326	17	11010111100101000	497	17	11010111010101100
156	17	11010111100000011	327	17	11010111100100000	498	17	11010111010011110
157	17	11010111011111011	328	17	11010111100011000	499	17	11010111010011100
158	17	11010111011011011	329	17	11010111100001000	500	17	11010111010010110
159	17	11010111011001011	330	17	11010111100000000	501	17	11010111010010100
160	17	11010111011000011	331	17	11010111011111000	502	17	11010111010001110
161	17	11010111010111011	332	17	11010111011011000	503	17	11010111010001100
162	17	11010111010110011	333	17	11010111011001000	504	17	11010111010000110
163	17	11010111010101011	334	17	11010111011000000	505	17	11010111010000100
164	17	11010111010011011	335	17	11010111010111000	506	17	11010111001111110
165	17	11010111010010011	336	17	11010111010110000	507	17	11010111001111100
166	17	11010111010001011	337	17	11010111010101000	508	17	11010111001110110
167	17	11010111010000011	338	17	11010111010011000	509	17	11010111001110100
168	17	11010111001111011	339	17	11010111010010000	510	17	11010111001101110
169	17	11010111001110011	340	17	11010111010001000	511	17	11010111001101100
170	17	11010111001110001	341	17	11010111010000000			

Table B.41 — AC_table, 512 symbols

symbol	no_ of_ bits	code	symbol	no_ of_ bits	code	symbol	no_ of_ bits	code
0	16	1000011100011000	171	16	1000011101100001	342	16	1000011101100000
1	16	1000011100011001	172	16	1000011110100001	343	15	100001110000000
2	16	1000011100011011	173	16	1000011111000001	344	16	1000011101101000
3	16	1000011100011101	174	16	1000011111100001	345	16	1000011110101000
4	16	1000011100011111	175	15	100001000100001	346	16	1000011111001000
5	16	1000011100100101	176	15	100001001100001	347	16	1000011111101000
6	16	1000011100100111	177	15	100001011000001	348	15	100001000101000
7	16	1000011100101101	178	15	100001011100001	349	15	100001001101000
8	16	1000011100101111	179	15	100001010100001	350	15	100001011001000
9	16	1000011100111101	180	15	100001010000001	351	15	100001011101000
10	16	1000011100111111	181	15	100001001000001	352	15	100001010101000
11	16	1000011101111101	182	15	100001000000001	353	15	100001010001000

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

12	16	1000011101111111	183	16	1000011110000001	354	15	100001001001000
13	16	1000011110111111	184	16	1000011101000001	355	15	100001000001000
14	16	1000011111011101	185	16	1000011101010001	356	16	1000011110001000
15	16	1000011111011111	186	16	1000011110010001	357	16	1000011101001000
16	16	1000011111111101	187	15	100001000010001	358	16	1000011101011000
17	16	1000011111111111	188	15	100001001010001	359	16	1000011110011000
18	15	100001000111101	189	15	100001010010001	360	15	100001000011000
19	15	100001000111111	190	15	100001010110001	361	15	100001001011000
20	15	100001001111101	191	15	100001011110001	362	15	100001010011000
21	15	100001001111111	192	15	100001011010001	363	15	100001010111000
22	15	100001011011101	193	15	100001001110001	364	15	100001011111000
23	15	100001011011111	194	15	100001000110001	365	15	100001011011000
24	15	100001011111101	195	16	1000011111110001	366	15	100001001111000
25	15	100001011111111	196	16	1000011111010001	367	15	100001000111000
26	15	100001010111111	197	16	1000011110110001	368	16	1000011111111000
27	15	100001010011101	198	16	1000011101110001	369	16	1000011111011000
28	15	100001010011111	199	16	1000011100110001	370	16	1000011110111000
29	15	100001001011111	200	15	100001110001001	371	16	1000011101111000
30	15	100001000011111	201	16	1000011100110101	372	16	1000011100111000
31	16	1000011110011111	202	16	1000011101110101	373	16	1000011100101000
32	16	1000011101011111	203	16	1000011110110101	374	16	1000011100100000
33	16	1000011101001111	204	16	1000011111010101	375	16	1000011100100010
34	16	1000011110001111	205	16	1000011111110101	376	16	1000011100101010
35	15	100001000001111	206	15	100001000110101	377	16	1000011100111010
36	15	100001001001111	207	15	100001001110101	378	16	1000011101111010
37	15	100001010001111	208	15	100001011010101	379	16	1000011110111010
38	15	100001010101111	209	15	100001011110101	380	16	1000011111011010
39	15	100001011101111	210	15	100001010110101	381	16	1000011111111010
40	15	100001011001111	211	15	100001010010101	382	15	100001000111010
41	15	100001001101111	212	15	100001001010101	383	15	100001001111010
42	15	100001000101111	213	15	100001000010101	384	15	100001011011010
43	16	1000011111101111	214	16	1000011110010101	385	15	100001011111010
44	16	1000011111001111	215	16	1000011101010101	386	15	100001010111010
45	16	1000011110101111	216	16	1000011101000101	387	15	100001010011010
46	16	1000011101101111	217	16	1000011110000101	388	15	100001001011010
47	15	100001110000111	218	15	100001000000101	389	15	100001000011010
48	16	1000011101100111	219	15	100001001000101	390	16	1000011110011010
49	16	1000011110100111	220	15	100001010000101	391	16	1000011101011010
50	16	1000011111000111	221	15	100001010100101	392	16	1000011101001010
51	16	1000011111100111	222	15	100001011100101	393	16	1000011110001010
52	15	100001000100111	223	15	100001011000101	394	15	100001000001010
53	15	100001001100111	224	15	100001001100101	395	15	100001001001010
54	15	100001011000111	225	15	100001000100101	396	15	100001010001010
55	15	100001011100111	226	16	1000011111100101	397	15	100001010101010
56	15	100001010100111	227	16	1000011111000101	398	15	100001011101010
57	15	100001010000111	228	16	1000011110100101	399	15	100001011001010
58	15	100001001000111	229	16	1000011101100101	400	15	100001001101010
59	15	100001000000111	230	15	100001110000101	401	15	100001000101010

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

60	16	1000011110000111	231	16	1000011101101101	402	16	1000011111101010
61	16	1000011101000111	232	16	1000011110101101	403	16	1000011111001010
62	16	1000011101010111	233	16	1000011111001101	404	16	1000011110101010
63	16	1000011110010111	234	16	1000011111101101	405	16	1000011101101010
64	15	100001000010111	235	15	100001000101101	406	15	100001110000010
65	15	100001001010111	236	15	100001001101101	407	16	1000011101100010
66	15	100001010010111	237	15	100001011001101	408	16	1000011110100010
67	15	100001010110111	238	15	100001011101101	409	16	1000011111000010
68	15	100001011110111	239	15	100001010101101	410	16	1000011111100010
69	15	100001011010111	240	15	100001010001101	411	15	100001000100010
70	15	100001001110111	241	15	100001001001101	412	15	100001001100010
71	15	100001000110111	242	15	100001000001101	413	15	100001011000010
72	16	1000011111110111	243	16	1000011110001101	414	15	100001011100010
73	16	1000011111010111	244	16	1000011101001101	415	15	100001010100010
74	16	1000011110110111	245	16	1000011101011101	416	15	100001010000010
75	16	1000011101110111	246	16	1000011110011101	417	15	100001001000010
76	16	1000011100110111	247	15	100001000011101	418	15	100001000000010
77	15	100001110001011	248	6	100000	419	16	1000011110000010
78	16	1000011100110011	249	15	100001001011101	420	16	1000011101000010
79	16	1000011101110011	250	15	100001010111101	421	16	1000011101010010
80	16	1000011110110011	251	7	1001110	422	16	1000011110010010
81	16	1000011111010011	252	6	100110	423	15	100001000010010
82	16	1000011111110011	253	5	10010	424	15	100001001010010
83	15	100001000110011	254	4	1010	425	15	100001010010010
84	15	100001001110011	255	2	11	426	15	100001010110010
85	15	100001011010011	256	16	1000011110111100	427	15	100001011110010
86	15	100001011110011	257	1	0	428	15	100001011010010
87	15	100001010110011	258	4	1011	429	15	100001001110010
88	15	100001010010011	259	6	100011	430	15	100001000110010
89	15	100001001010011	260	6	100010	431	16	1000011111110010
90	15	100001000010011	261	7	1001111	432	16	1000011111010010
91	16	1000011110010011	262	16	1000011110111101	433	16	1000011110110010
92	16	1000011101010011	263	8	10000110	434	16	1000011101110010
93	16	1000011101000011	264	15	100001010111100	435	16	1000011100110010
94	16	1000011110000011	265	15	100001001011100	436	15	100001110001010
95	15	100001000000011	266	15	100001000011100	437	16	1000011100110110
96	15	100001001000011	267	16	1000011110011100	438	16	1000011101110110
97	15	100001010000011	268	16	1000011101011100	439	16	1000011110110110
98	15	100001010100011	269	16	1000011101001100	440	16	1000011111010110
99	15	100001011100011	270	16	1000011110001100	441	16	1000011111110110
100	15	100001011000011	271	15	100001000001100	442	15	100001000110110
101	15	100001001100011	272	15	100001001001100	443	15	100001001110110
102	15	100001000100011	273	15	100001010001100	444	15	100001011010110
103	16	1000011111100011	274	15	100001010101100	445	15	100001011110110
104	16	1000011111000011	275	15	100001011101100	446	15	100001010110110
105	16	1000011110100011	276	15	100001011001100	447	15	100001010010110
106	16	1000011101100011	277	15	100001001101100	448	15	100001001010110
107	15	100001110000011	278	15	100001000101100	449	15	100001000010110

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

108	16	1000011101101011	279	16	1000011111101100	450	16	1000011110010110
109	16	1000011110101011	280	16	1000011111001100	451	16	1000011101010110
110	16	1000011111001011	281	16	1000011110101100	452	16	1000011101000110
111	16	1000011111101011	282	16	1000011101101100	453	16	1000011110000110
112	15	100001000101011	283	15	100001110000100	454	15	100001000000110
113	15	100001001101011	284	16	1000011101100100	455	15	100001001000110
114	15	100001011001011	285	16	1000011110100100	456	15	100001010000110
115	15	100001011101011	286	16	1000011111000100	457	15	100001010100110
116	15	100001010101011	287	16	1000011111100100	458	15	100001011100110
117	15	100001010001011	288	15	100001000100100	459	15	100001011000110
118	15	100001001001011	289	15	100001001100100	460	15	100001001100110
119	15	100001000001011	290	15	100001011000100	461	15	100001000100110
120	16	1000011110001011	291	15	100001011100100	462	16	1000011111100110
121	16	1000011101001011	292	15	100001010100100	463	16	1000011111000110
122	16	1000011101011011	293	15	100001010000100	464	16	1000011110100110
123	16	1000011110011011	294	15	100001001000100	465	16	1000011101100110
124	15	100001000011011	295	15	100001000000100	466	15	100001110000110
125	15	100001001011011	296	16	1000011110000100	467	16	1000011101101110
126	15	100001010011011	297	16	1000011101000100	468	16	1000011110101110
127	15	100001010111011	298	16	1000011101010100	469	16	1000011111001110
128	15	100001011111011	299	16	1000011110010100	470	16	1000011111101110
129	15	100001011011011	300	15	100001000010100	471	15	100001000101110
130	15	100001001111011	301	15	100001001010100	472	15	100001001101110
131	15	100001000111011	302	15	100001010010100	473	15	100001011001110
132	16	1000011111111011	303	15	100001010110100	474	15	100001011101110
133	16	1000011111011011	304	15	100001011110100	475	15	100001010101110
134	16	1000011110111011	305	15	100001011010100	476	15	100001010001110
135	16	1000011101111011	306	15	100001001110100	477	15	100001001001110
136	16	1000011100111011	307	15	100001000110100	478	15	100001000001110
137	16	1000011100101011	308	16	1000011111110100	479	16	1000011110001110
138	16	1000011100100011	309	16	1000011111010100	480	16	1000011101001110
139	16	1000011100100001	310	16	1000011110110100	481	16	1000011101011110
140	16	1000011100101001	311	16	1000011101110100	482	16	1000011110011110
141	16	1000011100111001	312	16	1000011100110100	483	15	100001000011110
142	16	1000011101111001	313	15	100001110001000	484	15	100001001011110
143	16	1000011110111001	314	16	1000011100110000	485	15	100001010011110
144	16	1000011111011001	315	16	1000011101110000	486	15	100001010011100
145	16	1000011111111001	316	16	1000011110110000	487	15	100001010111110
146	15	100001000111001	317	16	1000011111010000	488	15	100001011111110
147	15	100001001111001	318	16	1000011111110000	489	15	100001011111100
148	15	100001011011001	319	15	100001000110000	490	15	100001011011110
149	15	100001011111001	320	15	100001001110000	491	15	100001011011100
150	15	100001010111001	321	15	100001011010000	492	15	100001001111110
151	15	100001010011001	322	15	100001011110000	493	15	100001001111100
152	15	100001001011001	323	15	100001010110000	494	15	100001000111110
153	15	100001000011001	324	15	100001010010000	495	15	100001000111100
154	16	1000011110011001	325	15	100001001010000	496	16	1000011111111110
155	16	1000011101011001	326	15	100001000010000	497	16	1000011111111100

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

156	16	1000011101001001	327	16	1000011110010000	498	16	1000011111011110
157	16	1000011110001001	328	16	1000011101010000	499	16	1000011111011100
158	15	100001000001001	329	16	1000011101000000	500	16	1000011110111110
159	15	100001001001001	330	16	1000011110000000	501	16	1000011101111110
160	15	100001010001001	331	15	100001000000000	502	16	1000011101111100
161	15	100001010101001	332	15	100001001000000	503	16	1000011100111110
162	15	100001011101001	333	15	100001010000000	504	16	1000011100111100
163	15	100001011001001	334	15	100001010100000	505	16	1000011100101110
164	15	100001001101001	335	15	100001011100000	506	16	1000011100101100
165	15	100001000101001	336	15	100001011000000	507	16	1000011100100110
166	16	1000011111101001	337	15	100001001100000	508	16	1000011100100100
167	16	1000011111001001	338	15	100001000100000	509	16	1000011100011110
168	16	1000011110101001	339	16	1000011111100000	510	16	1000011100011100
169	16	1000011101101001	340	16	1000011111000000	511	16	1000011100011010
170	15	100001110000001	341	16	1000011110100000			

B.1.8 Shape decoding for still texture object

USInt sto_SI_bab_type_prob_even[1]={44651}

USInt sto_enh_odd_prob0[256]={

65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 25682, 29649, 83, 1096, 28876, 19253, 2320, 292, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 20597, 40071, 1, 1476, 6971, 9466, 4033, 314, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 34822, 58944, 1049, 888, 47296, 38677, 478, 14, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 21384, 35003, 26, 7, 9600, 13044, 44, 1, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 66, 13653, 1, 1, 25105, 19008, 285, 31, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1057, 27888, 1, 2521, 4252, 15367, 1, 751, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 290, 17476, 1, 1, 8625, 20056, 121, 3, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1960, 15984, 1, 1, 4747, 16480, 110, 9, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 87, 140, 1, 55, 1205, 1864, 1, 1, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 549, 2823, 1, 1, 1902, 782, 1285, 130, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 290, 6665, 1, 1, 15984, 8656, 1, 244, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 63, 2114, 1, 3, 2138, 1560, 69, 1, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1, 3178, 1, 218, 160, 66, 1, 2, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 35, 7047, 366, 180, 34, 113, 1, 9, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 837, 15124, 1, 1, 5571, 3236, 1, 2, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 14, 1538, 1, 14, 31, 115, 1, 1};

USInt sto_enh_even_prob0[256]={

65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536,25542, 38525, 48333, 53517, 26541, 23214, 61559, 59853,65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 7151, 31006, 33480, 29879, 2609, 4142, 16384, 11884, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 44817, 65535, 56097, 64984, 40735, 47710, 43386, 52046, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 17545, 29626, 37308, 42263, 4196, 13552, 7199, 7230, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1, 1, 1, 39322, 13578, 17416, 29218, 31831, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1311, 65535, 32768, 7282, 1, 3048, 25206, 19935, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1, 32768, 17873, 61861, 3417, 14895, 4541, 5293, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 4819, 32768, 39950, 43523, 1148, 4021, 12072, 5436, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1, 1, 134, 1, 1, 55, 5461, 2849, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 24, 5243, 590, 1079, 86, 95, 14564, 7159, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 328, 1, 2564, 14919, 21845, 1, 9362, 15880, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 43, 362, 150, 1179, 752, 529, 683, 331, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1, 1, 700, 862, 25, 24, 1317, 558, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 1, 1, 1, 172, 2, 4, 793, 342, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536,572, 1,

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1928, 43080, 3337, 1680, 1401, 2131, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536,18, 1, 366, 7456, 8, 18, 40, 8};

USInt sto_enh_odd_prob1[256]={

65536, 65536, 54236, 62309, 65536, 65536, 39146, 32459, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 3293, 20806, 65536, 65536, 20132, 22080, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 62837, 40750, 65536, 65536, 36528, 20960, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 46875, 21025, 65536, 65536, 35747, 30778, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 63956, 52735, 65536, 65536, 32974, 6233, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 39449, 40885, 65536, 65536, 23406, 10898, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 62156, 12655, 65536, 65536, 23973, 3451, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 31892, 3756, 65536, 65536, 24045, 14281, 0, 0, 0, 0, 0, 0, 0, 0, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 51966, 33057, 1, 6782, 62238, 32046, 5919, 418, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 64435, 18941, 1224, 7203, 52134, 4674, 6753, 1113, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 62703, 34133, 16705, 28007, 31477, 15453, 21558, 6073, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 61951, 29954, 4826, 6481, 12288, 2410, 4466, 228, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 1789, 23069, 726, 7470, 30386, 26721, 9811, 1446, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 45333, 28672, 21363, 26870, 41125, 9455, 25752, 12372, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 46324, 37071, 4994, 5136, 18879, 28687, 9330, 366, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 31745, 26116, 346, 397, 2900, 13830, 361, 8};

USInt sto_enh_even_prob1[256]={

65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536,0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 0, 0, 0, 0, 0, 0, 0, 0, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 65536, 0, 0, 0, 0, 0, 0, 0, 0, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 54018, 8704, 1, 903, 61648, 31196, 327, 575, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 64400, 36956, 1673, 9758, 52289, 4361, 659, 1433, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 56482, 65535, 1, 1, 11905, 3034, 1, 1, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 61103, 56650, 925, 8814, 11845, 2075, 828, 223, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 4161, 1, 494, 5041, 52508, 32195, 11005, 2463, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 63286, 32768, 39133, 49486, 53351, 8541, 37603, 15011, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 18805, 13107, 1039, 1214, 5060, 21845, 3830, 387, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 65537, 31654, 32951, 490, 1496, 2535, 11699, 328, 13};

B.2 Arithmetic Decoding

B.2.1 Aritmetic decoding for still texture object

To fully initialize the decoder, the function ac_decoder_init is called followed by ac_model_init respectively:

```
void ac_decoder_init (ac_decoder *acd) {
    int i, t;
    acd->bits_to_go = 0;
    acd->total_bits = 0;
    acd->value = 0;
    for (i=1; i<=Code_value_bits; i++)  {
        acd->value = 2*acd->value + input_bit(acd);
    }
    acd->low = 0;
    acd->high = Top_value;
    return;
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
void ac_model_init (ac_model *acm, int nsym) {
    int i;

    acm->nsym = nsym;

    acm->freq = (unsigned short *) malloc (nsym*sizeof (unsigned short));
    check (!acm->freq, "arithmetic coder model allocation failure");
    acm->cfreq = (unsigned short *) calloc (nsym+1, sizeof (unsigned short));
    check (!acm->cfreq, "arithmetic coder model allocation failure");

    for (i=0; i<acm->nsym; i++) {
        acm->freq[i] = 1;
        acm->cfreq[i] = acm->nsym - i;
    }
    acm->cfreq[acm->nsym] = 0;

    return;
}
```

The acd is structures which contains the decoding variables and whose addresses act as handles for the decoded symbol/bitstreams. The fields bits_to_go, buffer, bitstream, and bitstream_len are used to manage the bits in memory. The low, high, and fbits fields describe the scaled range corresponding to the symbols which have been decoded. The value field contains the currently seen code value inside the range. The total_bits field contains the total number of bits encoded or used for decoding so far. The values Code_value_bits and Top_value describe the maximum number of bits and the maximum size of a coded value respectively. The ac_model structure contains the variables used for that particular probability model and it's address acts as a handle. The nsym field contains the number of symbols in the symbol set, the freq field contains the table of frequency counts for each of the nsym symbols, and the cfreq field contains the cumulative frequency count derived from freq.

The bits are read from the bitstream using the function:

```
static int input_bit (ac_decoder *acd) {
    int t;
    unsigned int tmp;

    if (acd->bits_to_go==0)  {
        acd->buffer =  ace->bitstream[ace->bitstream_len++];
        acd->bits_to_go = 8;
    }

    t = acd->buffer & 0x080;
    acd->buffer <<= 1;
    acd->buffer &=  0x0ff;
    acd->total_bits += 1;
    acd->bits_to_go -= 1;
    t = t >> 7;

    return t;
}
```

The decoding process has four main steps. The first step is to decode the symbol based on the current state of the probability model (frequency counts) and the current code value (value) which is used to represent (and is a member of) the current range. The second step is to get the new range. The third step is to rescale the range and

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

simultaneously load in new code value bits. The fourth step is to update the model. To decode symbols, the following function is called:

```
int ac_decode_symbol (ac_decoder *acd, ac_model *acm) {
    long range;
    int cum;
    int sym;

    range = (long)(acd->high-acd->low)+1;

    /*--- decode symbol ---*/
    cum = (((long)(acd->value-acd->low)+1)*(int)(acm->cfreq[0])-1)/range;
    for (sym = 0; (int)acm->cfreq[sym+1]>cum; sym++)
        /* do nothing */ ;

    check (sym<0||sym>=acm->nsym, "symbol out of range");

    /*--- Get new range ---*/
    acd->high = acd->low + (range*(int)(acm->cfreq[sym]))/(int)(acm->cfreq[0])-1;
    acd->low = acd->low +  (range*(int)(acm->cfreq[sym+1]))/(int)(acm->cfreq[0]);

    /*--- rescale and load new code value bits ---*/
    for (;;)  {
        if (acd->high<Half)  {
            /* do nothing */
        }  else if (acd->low>=Half)  {
            acd->value -= Half;
            acd->low -= Half;
            acd->high -= Half;
        }  else if (acd->low>=First_qtr && acd->high<Third_qtr)  {
            acd->value -= First_qtr;
            acd->low -= First_qtr;
            acd->high -= First_qtr;
        }  else
            break;
        acd->low = 2*acd->low;
        acd->high = 2*acd->high+1;
        acd->value = 2*acd->value + input_bit(acd);
    }

    /*--- Update probability model ---*/
    update_model (acm, sym);

    return sym;
}
```

The update of the probability model used in the decoding of the symbols is shown in the following function:

```
static void update_model (ac_model *acm, int sym)
{
    int i, Max_frequency =127;
```

--`,,--,,,,-,,,,,,--- © ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    if (acm->cfreq[0]==Max_frequency) {
        int cum = 0;
        acm->cfreq[acm->nsym] = 0;
        for (i = acm->nsym-1; i>=0; i--) {
            acm->freq[i] = ((int)acm->freq[i] + 1) / 2;
            cum += acm->freq[i];
            acm->cfreq[i] = cum;
        }
    }

    acm->freq[sym] += 1;
    for (i=sym; i>=0; i--)
        acm->cfreq[i] += 1;

    return;
}
```

This function updates the frequency counts based on the symbol just decoded. It also makes sure that the maximum frequency allowed is not exceeded. This is done by rescaling all frequency counts by 2.

B.2.2 Arithmetic decoding for shape decoding

B.2.2.1 Structures and Typedefs

```
typedef void Void;
typedef int Int;
typedef unsigned short int USInt;
#define  CODE_BIT 32
#define  HALF  ((unsigned) 1 << (CODE_BITS-1))
#define  QUARTER  (1 << (CODE_BITS-2))
struct arcodec {
  UInt L; /* lower bound */
  UInt R; /* code range */
  UInt V; /* current code value */
  UInt arpipe;
  Int bits_to_follow; /* follow bit count */
  Int first_bit;
  Int nzeros;
  Int nonzero;
  Int nzerosf;
  Int extrabits;
};
typedef struct arcodec ArCoder;
typedef struct arcodec ArDecoder;
#define MAXHEADING 3
#define MAXMIDDLE 10
#define MAXTRAILING 2
```

B.2.2.2 Decoder Source

```
Void StartArDecoder(ArDecoder *decoder, Bitstream *bitstream) {
  Int i,j;
  decoder->V = 0;
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  decoder->nzerosf = MAXHEADING;
  decoder->extrabits = 0;
  for (i = 1; i<CODE_BITS; i++) {
    j=BitstreamLookBit(bitstream,i+decoder->extrabits);
    decoder->V += decoder->V + j;
    if (j == 0) {
      decoder->nzerosf--;
      if (decoder->nzerosf == 0) {
        decoder->extrabits++;
        decoder->nzerosf = MAXMIDDLE;
      }
    }
    else
      decoder->nzerosf = MAXMIDDLE;
  }
  decoder->L = 0;
  decoder->R = HALF - 1;
  decoder->bits_to_follow = 0;
  decoder->arpipe = decoder->V;
  decoder->nzeros = MAXHEADING;
  decoder->nonzero = 0;
}
Void StopArDecoder(ArDecoder *decoder, Bitstream *bitstream) {
  Int a = decoder->L >> (CODE_BITS-3);
  Int b = (decoder->R + decoder->L) >> (CODE_BITS-3);
  Int nbits,i;
  if (b == 0)
    b = 8;
  if (b-a >= 4 || (b-a == 3 && a&1))
    nbits = 2;
  else
    nbits = 3;
  for (i = 1; i <= nbits-1; i++)
    AddNextInputBit(bitstream, decoder);
  if (decoder->nzeros < MAXMIDDLE-MAXTRAILING || decoder->nonzero == 0)
    BitstreamFlushBits(bitstream,1);
}
Void AddNextInputBit(Bitstream *bitstream, ArDecoder *decoder) {
  Int i;
  if (((decoder->arpipe >> (CODE_BITS-2))&1) == 0) {
    decoder->nzeros--;
    if (decoder->nzeros == 0) {
      BitstreamFlushBits(bitstream,1);
      decoder->extrabits--;
      decoder->nzeros = MAXMIDDLE;
      decoder->nonzero = 1;
    }
  }
  else {
    decoder->nzeros = MAXMIDDLE;
    decoder->nonzero = 1;
  }
  BitstreamFlushBits(bitstream,1);
  i = (Int)BitstreamLookBit(bitstream, CODE_BITS-1+decoder->extrabits);
  decoder->V += decoder->V + i;
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    decoder->arpipe += decoder->arpipe + i;
    if (i == 0) {
      decoder->nzerosf--;
      if (decoder->nzerosf == 0) {
        decoder->nzerosf = MAXMIDDLE;
        decoder->extrabits++;
      }
    }
    else
      decoder->nzerosf = MAXMIDDLE;
  }
  Int ArDecodeSymbol(USInt c0, ArDecoder *decoder, Bitstream *bitstream ) {
    Int bit;
    Int c1 = (1<<16) - c0;
    Int LPS = c0 > c1;
    Int cLPS = LPS ? c1 : c0;
    unsigned long rLPS;
    rLPS = ((decoder->R) >> 16) * cLPS;
    if ((decoder->V - decoder->L) >= (decoder->R - rLPS)) {
      bit = LPS;
      decoder->L += decoder->R - rLPS;
      decoder->R = rLPS;
    }
    else {
      bit = (1-LPS);
      decoder->R -= rLPS;
    }
    DECODE_RENORMALISE(decoder,bitstream);
    return(bit);
  }
  Void DECODE_RENORMALISE(ArDecoder *decoder,  Bitstream *bitstream) {
    while (decoder->R < QUARTER) {
      if (decoder->L >= HALF) {
        decoder->V -= HALF;
        decoder->L -= HALF;
        decoder->bits_to_follow = 0;
      }
      else
        if (decoder->L + decoder->R <= HALF)
          decoder->bits_to_follow = 0;
        else{
          decoder->V -= QUARTER;
          decoder->L -= QUARTER;
          (decoder->bits_to_follow)++;
        }
      decoder->L += decoder->L;
      decoder->R += decoder->R;
      AddNextInputBit(bitstream, decoder);
    }
  }
```

- BitstreamLookBit(bitstream,nbits) : Looks nbits ahead in the bitstream beginning from the current position in the bitstream and returns the bit.
- BitstreamFlushBits(bitstream,nbits) : Moves the current bitstream position forward by nbits.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

- The parameter c0 (used in ArDecodeSymbol()) is taken directly from the probability tables of USint inter_prob or Usint intra_prob in Table B.32. That is, for the pixel to be coded/decoded, c0 is the probability than this pixel is equal to zero. The value of c0 depends on the context number of the given pixel to be decoded.

B.2.3 FBA Object Decoding

In FBA decoder, a symbol is decoded by using a specific model based on the syntax and by calling the following procedure which is specified in C.

```
static long low, high, code_value, bit, length, sacindex, cum, zerorun=0;

int aa_decode(int cumul_freq[ ])
{
  length = high - low + 1;
  cum = (-1 + (code_value - low + 1) * cumul_freq[0]) / length;
  for (sacindex = 1; cumul_freq[sacindex] > cum; sacindex++);
  high = low - 1 + (length * cumul_freq[sacindex-1]) / cumul_freq[0];
  low += (length * cumul_freq[sacindex]) / cumul_freq[0];

  for ( ; ; ) {
    if (high < q2) ;
    else if (low >= q2) {
      code_value -= q2;
      low -= q2;
      high -= q2;
    }
    else if (low >= q1 && high < q3) {
      code_value -= q1;
      low -= q1;
      high -= q1;
    }
    else {
      break;
    }
    low *= 2;
    high = 2*high + 1;
    bit_out_psc_layer();
    code_value = 2*code_value + bit;
    used_bits++;
  }
  return (sacindex-1);
}

void bit_out_psc_layer()
{
  bit = getbits(1);
}
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Again the model is specified through `cumul_freq[ ]`. The decoded symbol is returned through its index in the model. The decoder is initialized to start decoding an arithmetic coded bitstream by calling the following procedure.

```
void decoder_reset( )
{
  int i;
  zerorun = 0;        /* clear consecutive zero's counter */
  code_value = 0;
  low = 0;
  high = top;
  for (i = 1;   i <= 16;   i++) {
    bit_out_psc_layer();
    code_value = 2 * code_value + bit;
  }
  used_bits = 0;
}
```

B.3 Variable length codes for the studio profile

B.3.1 Macroblock type

The properties of the macroblock are determined by the macroblock type VLC according to these tables.

Table B.42 — Variable length codes for macroblock_type in I-VOPs

macroblock_type VLC code	macroblock_quant	macroblock_motion_forward	macroblock_motion_backward	macroblock_pattern	macroblock_intra	Description
1	0	0	0	0	1	Intra
01	1	0	0	0	1	Intra, Quant

Table B.43 — Variable length codes for macroblock_type in P-VOPs

macroblock_type VLC code	macroblock_quant	macroblock_motion_forward	macroblock_motion_backward	macroblock_pattern	macroblock_intra	Description
1	0	1	0	1	0	MC, Coded
01	0	0	0	1	0	No MC, Coded
001	0	1	0	0	0	MC, Not Coded
0001 1	0	0	0	0	1	Intra
0001 0	1	1	0	1	0	MC, Coded, Quant
0000 1	1	0	0	1	0	No MC, Coded, Quant
0000 01	1	0	0	0	1	Intra, Quant

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

B.3.2 DCT coefficients

Table B.44 — Variable length codes for dct_dc_size_luminance

Variable length code	dct_dc_size_luminance
001110	0
00110	1
0000	2
0010	3
111	4
101	5
011	6
010	7
100	8
110	9
0001	10
0011110	11
00111110	12
001111110	13
0011111110	14
00111111110	15
001111111110	16
0011111111110	17
0011111111111	18

Table B.45 — Variable length codes for dct_dc_size_chrominance

Variable length code	dct_dc_size_chrominance
0000	0
0010	1
111	2
101	3
011	4
010	5
100	6
110	7
0001	8
00110	9
001110	10
0011110	11
00111110	12
001111110	13
0011111110	14
00111111110	15
001111111110	16
0011111111110	17
0011111111111	18

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.46 — Differential DC additional codes

Additional code	Differential DC	Size
000000000000000000 to 011111111111111111 *	-262143 to -131072	18
00000000000000000 to 01111111111111111 *	-131071 to -65536	17
0000000000000000 to 0111111111111111 *	-65535 to -32768	16
000000000000000 to 011111111111111 *	-32767 to -16384	15
00000000000000 to 01111111111111 *	-16383 to -8192	14
0000000000000 to 0111111111111 *	-8191 to -4096	13
000000000000 to 011111111111 *	-4095 to -2048	12
00000000000 to 01111111111 *	-2047 to -1024	11
0000000000 to 0111111111 *	-1023 to -512	10
000000000 to 011111111 *	-511 to -256	9
00000000 to 01111111	-255 to -128	8
0000000 to 0111111	-127 to -64	7
000000 to 011111	-63 to -32	6
00000 to 01111	-31 to -16	5
0000 to 0111	-15 to -8	4
000 to 011	-7 to -4	3
00 to 01	-3 to -2	2
0	-1	1
	0	0
1	1	1
10 to 11	2 to 3	2
100 to 111	4 to 7	3
1000 to 1111	8 to 15	4
10000 to 11111	16 to 31	5
100000 to 111111	32 to 63	6
1000000 to 1111111	64 to 127	7
10000000 to 11111111	128 to 255	8
100000000 to 111111111 *	256 to 511	9
1000000000 to 1111111111 *	512 to 1023	10
10000000000 to 11111111111 *	1024 to 2047	11
100000000000 to 111111111111 *	2048 to 4095	12
1000000000000 to 1111111111111 *	4096 to 8191	13
10000000000000 to 11111111111111 *	8192 to 16383	14
100000000000000 to 111111111111111 *	16384 to 32767	15
1000000000000000 to 1111111111111111 *	32768 to 65535	16
10000000000000000 to 11111111111111111 *	65536 to 131071	17
100000000000000000 to 111111111111111111 *	131072 to 262143	18

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

In cases where dct_dc_size is greater than 8, marked '*' in Table B46, a marker bit is inserted after the dct_dc_additional_code to prevent start code emulations.

Table B.47 — Additional codes for group No.1~6 (zero run length)

Size	Additional code	zero run length
0	---	1
1	0 to 1	2 to 3
2	00 to 11	4 to 7
3	000 to 111	8 to 15
4	0000 to 1111	16 to 31
5	00000 to 11111	32 to 63

Table B.48 — Additional codes for group No.7~12 (zero run length and +/-1 level)

Size	Additional code	zero run length	level(+1/-1)
6	000000 to 111110	32 to 63	-1
5	00000 to 11110	16 to 31	-1
4	0000 to 1110	8 to 15	-1
3	000 to 110	4 to 7	-1
2	00 to 10	2 to 3	-1
1	0	1	-1
0	---	---	---
1	1	1	+1
2	01 to 11	2 to 3	+1
3	001 to 111	4 to 7	+1
4	0001 to 1111	8 to 15	+1
5	00001 to 11111	16 to 31	+1
6	000001 to 111111	32 to 63	+1

Table B.49 — Additional codes for group No.13~20 (level value)

Size	Additional code	level value
8	00000000 to 01111111	-255 to -128
7	0000000 to 0111111	-127 to -64
6	000000 to 011111	-63 to -32
5	00000 to 01111	-31 to -16
4	0000 to 0111	-15 to -8
3	000 to 011	-7 to -4
2	00 to 01	-3 to -2
1	0	-1
0	---	---
1	1	1
2	10 to 11	2 to 3
3	100 to 111	4 to 7
4	1000 to 1111	8 to 15
5	10000 to 11111	16 to 31
6	100000 to 111111	32 to 63
7	1000000 to 1111111	64 to 127
8	10000000 to 11111111	128 to 255

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.50 — VLC Table T[0] for intra macroblock

symbol group	variable length code
0	0101
1	0100
2	0000101
3	000001001
4	00000100001
5	000001000001
6	0000010000001
7	0011
8	00011
9	000101
10	0000100
11	0000011
12	00000101
13	11
14	101
15	100
16	011
17	0010
18	000100
19	000011
20	0000010001
21	0000010000000

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.51 — VLC Table T[1] for intra macroblock

symbol group	variable length code
0	---
1	---
2	---
3	---
4	---
5	---
6	---
7	---
8	---
9	---
10	---
11	---
12	---
13	---
14	1
15	01
16	001
17	0001
18	00001
19	0000011
20	00000101
21	00000100

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.52 — VLC Table T[2] for intra macroblock

symbol group	variable length code
0	101
1	00011
2	00010
3	0000011
4	000001001
5	00000100000011
6	00000100000010
7	100
8	011
9	0011
10	0010
11	000011
12	0000010001
13	11
14	010
15	000010
16	00000101
17	00000100001
18	0000010000011
19	00000100000001
20	000001000000001
21	0000010000010

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.53 — VLC Table T[3] for intra macroblock

symbol group	variable length code
0	00101
1	0101
2	00100
3	000011
4	000001001
5	0000010000011
6	0000010000010
7	011
8	0100
9	0011
10	00011
11	00000101
12	0000010000001
13	11
14	10
15	00010
16	000010
17	0000011
18	0000010001
19	000001000011
20	0000010000000
21	000001000010

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.54 — VLC Table T[4] for intra macroblock

symbol group	variable length code
0	0000101
1	0011
2	00011
3	0000100
4	000001001
5	0000010000011
6	00000100000001
7	011
8	0010
9	000101
10	0000011
11	0000010001
12	000001000000001
13	11
14	10
15	010
16	000100
17	000011
18	00000101
19	00000100001
20	0000010000010
21	0000010000001

--`,,`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.55 — VLC Table T[5] for intra macroblock

symbol group	variable length code
0	0000010011
1	00011
2	0000101
3	0000010010
4	000001000011
5	0000010000011
6	0000010000010
7	00010
8	0000100
9	00000101
10	00000100011
11	0000010000001
12	00000100000001
13	11
14	10
15	01
16	001
17	000011
18	0000011
19	00000100010
20	000001000000001
21	000001000010

-`-`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.56 — VLC Table T[6] for intra macroblock

symbol group	variable length code
0	00000100011
1	0001
2	00000111
3	0000010011
4	00000100010
5	00000100000011
6	00000100000010
7	000011
8	00000110
9	0000010010
10	000001000011
11	00000100000001
12	000001000000001
13	011
14	010
15	11
16	10
17	001
18	000010
19	00000101
20	000001000010
21	000001000001

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.57 — VLC Table T[7] for intra macroblock

symbol group	variable length code
0	000001011
1	00011
2	00000111
3	0000111
4	0000110
5	00000100011
6	000001000001
7	0000101
8	00000110
9	000001010
10	0000010011
11	00000100010
12	000001000000
13	0011
14	0010
15	11
16	10
17	01
18	00010
19	0000100
20	0000010010
21	00000100001

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.58 — VLC Table T[8] for intra macroblock

symbol group	variable length code
0	0000010101
1	000011
2	0000010100
3	00000100011
4	00000111
5	000001000011
6	0000010000001
7	00000110
8	000001011
9	0000010011
10	0000010010
11	000001000010
12	0000010000000
13	0001
14	011
15	010
16	11
17	10
18	001
19	000010
20	00000100010
21	000001000001

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.59 — VLC Table T[9] for intra macroblock

symbol group	variable length code
0	000001000011
1	000101
2	00000111
3	000100
4	000011
5	0000010011
6	000001000010
7	0000101
8	0000100
9	00000110
10	0000010010
11	000001000001
12	000001000000
13	00011
14	0011
15	011
16	010
17	11
18	10
19	0010
20	00000101
21	0000010001

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.60 — VLC Table T[10] for intra macroblock

symbol group	variable length code
0	0000010000011
1	0000101
2	00000111
3	0011
4	00000100001
5	0000010000010
6	0000010000001
7	0000100
8	00000110
9	000001011
10	000001010
11	0000010001
12	0000010000000
13	00011
14	00010
15	0010
16	011
17	010
18	11
19	10
20	000011
21	000001001

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.61 — VLC Table T[11] for intra macroblock

symbol group	variable length code
0	0000010011
1	00011
2	000011
3	000001101
4	000001100
5	00000100001
6	00000100000
7	00010
8	000010
9	00000111
10	000001011
11	0000010010
12	0000010001
13	101
14	100
15	0101
16	0100
17	0011
18	0010
19	011
20	11
21	000001010

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.62 — VLC Table T[0] for inter macroblock

symbol group	variable length code
0	---
1	00101
2	00100
3	0000011
4	000001001
5	00000100001
6	0000010000010
7	101
8	100
9	0101
10	0100
11	00011
12	00010
13	11
14	011
15	0011
16	000011
17	000010
18	00000101
19	0000010001
20	0000010000001
21	0000010000000

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.63 — VLC Table T[1] for inter macroblock

symbol group	variable length code
0	---
1	---
2	---
3	---
4	---
5	---
6	---
7	---
8	---
9	---
10	---
11	---
12	---
13	---
14	1
15	01
16	001
17	0001
18	00001
19	0000011
20	00000101
21	00000100

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.64 — VLC Table T[2] for inter macroblock

symbol group	variable length code
0	101
1	00101
2	00100
3	000011
4	0000011
5	0000010001
6	0000010000011
7	100
8	011
9	010
10	0011
11	00011
12	00000101
13	11
14	00010
15	000010
16	000001001
17	000001000011
18	0000010000010
19	0000010000001
20	0000010000000
21	000001000010

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.65 — VLC Table T[3] for inter macroblock

symbol group	variable length code
0	0101
1	0100
2	00011
3	000101
4	000100
5	0000010001
6	000001000001
7	101
8	100
9	0011
10	0010
11	000011
12	00000101
13	11
14	011
15	000010
16	0000011
17	000001001
18	00000100001
19	00000100000001
20	000001000000001
21	0000010000001

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.66 — VLC Table T[4] for inter macroblock

symbol group	variable length code
0	000101
1	0101
2	00101
3	000100
4	000011
5	00000100001
6	0000010000001
7	011
8	0100
9	0011
10	00100
11	0000011
12	000001001
13	11
14	10
15	00011
16	000010
17	00000101
18	0000010001
19	00000100000001
20	000001000000001
21	000001000001

--`,,`-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2004 – All rights reserved
Not for Resale

Table B.67 — VLC Table T[5] for inter macroblock

symbol group	variable length code
0	000101
1	011
2	00101
3	0000101
4	0000100
5	000001000011
6	0000010000001
7	0011
8	00100
9	000100
10	0000011
11	00000101
12	0000010001
13	11
14	10
15	010
16	00011
17	000011
18	000001001
19	000001000010
20	0000010000000
21	000001000001

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.68 — VLC Table T[6] for inter macroblock

symbol group	variable length code
0	000101
1	0011
2	000100
3	0000101
4	00000101
5	000001000011
6	0000010000001
7	00101
8	00100
9	000011
10	0000100
11	000001001
12	0000010001
13	011
14	11
15	10
16	010
17	00011
18	0000011
19	000001000010
20	0000010000000
21	000001000001

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.69 — VLC Table T[7] for inter macroblock

symbol group	variable length code
0	0000101
1	0011
2	000101
3	000001011
4	0000010011
5	00000100011
6	000001000001
7	00011
8	000100
9	0000100
10	0000011
11	000001010
12	00000100010
13	011
14	010
15	11
16	10
17	0010
18	000011
19	0000010010
20	000001000000
21	00000100001

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.70 — VLC Table T[8] for inter macroblock

symbol group	variable length code
0	000101
1	0011
2	0000111
3	0000010001
4	000001000011
5	0000010000001
6	0000010000000
7	000100
8	0000110
9	0000101
10	0000100
11	00000101
12	000001001
13	0010
14	011
15	010
16	11
17	10
18	00011
19	0000011
20	000001000010
21	000001000001

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.71 — VLC Table T[9] for inter macroblock

symbol group	variable length code
0	000101
1	0011
2	0000101
3	0000100
4	00000100001
5	0000010000001
6	0000010000000
7	0010
8	000100
9	000011
10	0000011
11	0000010001
12	000001001
13	111
14	110
15	101
16	100
17	011
18	010
19	00011
20	00000101
21	000001000001

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.72 — VLC Table T[10] for inter macroblock

symbol group	variable length code
0	00000100011
1	0011
2	0000011
3	00000100010
4	0000010000011
5	0000010000010
6	0000010000001
7	00011
8	000011
9	00000101
10	000001001
11	000001000011
12	0000010000000
13	101
14	100
15	11
16	011
17	0010
18	010
19	00010
20	000010
21	000001000010

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.73 — VLC Table T[11] for inter macroblock

symbol group	variable length code
0	0000010101
1	0001
2	000011
3	0000010100
4	000001000011
5	0000010000001
6	00000100000001
7	000001011
8	0000010011
9	0000010010
10	00000100011
11	00000100010
12	000001000000001
13	001
14	11
15	10
16	01
17	00000111
18	00000110
19	000010
20	000001000010
21	000001000001

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.3.3 Shape Coding

Table B.74 — Meaning of shape mode

Index	Shape mode
0	"MVs==0 && No Update"
1	"MVs!=0 && No Update"
2	transparent
3	opaque
4	“HHC”

Table B.75 — bab_type for I-VOP

Index	(2)	(3)	(4)
0	1	001	01
1	001	01	1
2	01	001	1

Table B.76 — bab_type for P-VOP

		bab_type in current VOP (n)				
		0	**1**	**2**	**3**	**4**
bab_type in previous VOP(n-1)	**0**	1	01	00001	000001	0001
	1	01	1	00001	000001	001
	2	0001	001	1	000001	01
	3	1	0001	000001	001	01
	4	011	001	0001	00001	1

Table B.77 — VLC table for MVs

MVs	Codes
0	0
±1	10s
±2	110s
±3	1110s
±4	11110s
±5	111110s
±6	1111110s
±7	11111110s
±8	111111110s
±9	1111111110s
±10	11111111110s
±11	111111111110s
±12	1111111111110s
±13	11111111111110s
±14	111111111111110s
±15	1111111111111110s
±16	11111111111111110s

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.78 — VLC table for MVs (Horizontal element is 0)

MVDs	Codes
±1	0s
±2	10s
±3	110s
±4	1110s
±5	11110s
±6	111110s
±7	1111110s
±8	11111110s
±9	111111110s
±10	1111111110s
±11	11111111110s
±12	111111111110s
±13	1111111111110s
±14	11111111111110s
±15	111111111111110s
±16	1111111111111110s
s: sign bit (if MVs is positive s="1", otherwise s="0").	

Table B.79 — VLC table for cbbp

cbbp index	code	cbbp index	code
1	1000	9	0000001
2	1010	10	1011
3	111	11	0111
4	1001	12	110
5	001	13	0101
6	000001	14	0110
7	0100	15	00001
8	0001		

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.80 — VLC table for clp

clp index	code	clp index	code	clp index	code
0	00111	86	00101111010010000001010	172	0010111101001000010011
1	01	87	10000111100101110	173	0010111101001000010010
2	100010	88	100000111110100	174	0010111101001000011010
3	1111	89	00101111010010000111110	175	100000010001001
4	00101001	90	0010111101001000010101	176	00101111011111
5	100000011000	91	0010111101001000000001	177	0010100001010011
6	100000110	92	001011110100100001	178	00101111010010000000111
7	1110	93	00101111010010000110110	179	00110110000101111
8	10000010	94	10000111100101011	180	100000011011110001
9	001011100101	95	00110110111101	181	00101111010010000011100 1
10	0010111001100	96	1000010101	182	0010111101001000011001
11	001101100010	97	001011110000000	183	00110110000101110
12	0011011100	98	00110110111100	184	00110110111011
13	001101101100	99	0010100001100	185	00110110000101101
14	0010111011	100	100001111000110	186	001101101101001010
15	1100	101	0010111101001000001001	187	001101101101000111
16	100000000	102	00110110000100010	188	0011011000001101
17	0010100001011	103	00101110011110	189	001101101101000100
18	0010111001101	104	10000001101110	190	001101100000100
19	10000100101010	105	00110110000100110	191	100000011010
20	10000011110110	106	00110110000101011	192	1011
21	001101101101001100	107	10000111100101100	193	001010001
22	001101100000011	108	001101101101001011	194	0011011000011
23	1000011110000	109	001101101101000000	195	1000010001
24	1000000010	110	0010111101001000000011	196	1000001111010
25	00110110111010	111	10000011111000	197	00101111010010000110111
26	1000011110011	112	1000000011	198	10000101100110
27	0010111101100	113	001011100111110	199	0011011010

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

28	1000000111	114	100001111000111	200	0011011011011
29	10000001101101	115	100000111110011	201	1000001111101111
30	0011011001	116	100000010001000	202	00110110000101100
31	10011	117	001011110100100000111111	203	001101100000010
32	001100	118	100000011011110011	204	00101111011110
33	001011110101	119	001011110100101	205	100000011011110000
34	100001001011001	120	0011011110	206	00110110111001
35	0010111101101	121	00101000011011	207	0010111010
36	0010111101000	122	1000001111101110	208	00101000001
37	00110110000100111	123	100001111000100	209	0011011000001011
38	001011110100100000111000	124	0010101000	210	001101101101001101
39	001010000111	125	100000011011111	211	00101000010100000
40	100001111000101	126	10000001001	212	10000111100101111
41	1000010010111011	127	10000110	213	00101111010010000001101
42	00101000010100001	128	1010	214	00101111010010000001011
43	10000111100101101	129	100001110	215	1000001111101101
44	1000001111101010	130	1000001110	216	10000100101111
45	001011110100100000011000	131	1000010100	217	00110110000101010
46	0010111101001001	132	0010111001000	218	00110110000101001
47	1000010110010	133	0011011000001100	219	1000001111101100
48	1000010111	134	10000101100111	220	100000111110010
49	100001001011010	135	1000010011	221	001101101101001001
50	00110110000000	136	10000100101011	222	001101101101001000
51	00110110110101	137	001011110000001	223	00101110001
52	0010111100010	138	001011110100100000000101	224	000
53	001101101101000101	139	100000010001011	225	001010101
54	1000011110010011	140	00101110011101	226	1000000100011
55	00110110111000	141	001011110100100000001000	227	0011011101
56	0010111111	142	1000011110010001	228	001010000100
57	00101000011010	143	0010111110	229	001011110100100000010000

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

58	001011100111111	144	100000111111	230	00101000010101
59	00110110000100101	145	00110110000101000	231	0010101001
60	10000111101	146	00101111010010000000110	232	100001011000
61	00101111010011	147	1000000110111101	233	001011110100100000001100
62	1000000101	148	001011110100100000111011	234	001101101101000011
63	100011	149	00101111010010000010001	235	1000001111101011
64	00101011	150	001101101101000010	236	10000011110111
65	0010111100011	151	1000011110010100	237	00110110000100100
66	100000010000	152	100000010001010	238	0011011000001010
67	0011011011111	153	0010100001010010	239	00101110000
68	100001001011000	154	001011110100100000111010	240	1101
69	00101111010010000010110	155	00110110000100011	241	1000011111
70	00110110000011110	156	100001001011100	242	1000010010100
71	0010111101110	157	001101101101001110	243	10000101101
72	10000001101100	158	100001001011011	244	001101100011
73	00101111010010000000100	159	10000100100	245	00110110000100000
74	00101111010010001	160	00101111001	246	00101111000001
75	001101101101001111	161	00101110011100	247	00101000000
76	00110110000100001	162	00110110000011111	248	10010
77	001011110100100000111101	163	10000111100101010	249	1000010000
78	001101101101000110	164	1000011110010010	250	100000111100
79	0010111001001	165	100000011011110010	251	100000011001
80	0010111100001	166	00101111010010000010111	252	00100
81	1000011110010000	167	0011011000001110	253	0011011111
82	1000010010111010	168	001011110100100000111100	254	0010110
83	001101101101000001	169	00101111010010000001111	255	0011010
84	00101111010010000001110	170	00101111010010000000000		
85	00101111010010000000010	171	00101111010010000010100		

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.81 — VLC table for lbp

lbp index	code	lbp index	code	lbp index	code
1	0111110110	86	0111101001110110101	171	01111101110100110011
2	01110101111	87	011101010101101100101	172	011101101100011
3	0111010011	88	01111010011110	173	01110101010101101101
4	01110101001	89	011110100110001100	174	0111011001111011
5	01111101000110111	90	011110100111011001 1	175	0111110000001101
6	0111110100100	91	011110100111011000	176	01110110010
7	011101101010	92	011101101100010	177	011100001011111
8	0111001010	93	011101010101011011 00100	178	0111001011011100
9	0111111011001010	94	01110110011111000	179	0111011001101100
10	0111011001110110	95	01110110011111010	180	011100001011110
11	01110110011111001	96	01110011	181	011111111101010011
12	01110101000	97	0111000011110	182	01111101110000011
13	01111000111101111	98	011101010101010	183	0111110000001100
14	011100001001	99	011111000000100	184	011110001110
15	011111111100	100	01110101110010	185	011100101100001
16	011110010	101	01111101110000000	186	011111101100101111
17	01111101000111	102	0111011001111000	187	0111011001111010
18	011101101100001	103	0111001011010101	188	011110001101
19	0111011001100001	104	01110010110110	189	01111101110000010
20	01110110110010	105	011110100110001101	190	0111010101100
21	011101010101011011 00110	106	01110010110111010	191	0111101001101
22	011101001001100101	107	011101010101011011 00010	192	100
23	0111010101010110111	108	011100101101011	193	011100100
24	0111101101	109	0111101001110110100	194	011100001100
25	011101001001101	110	011101010101011000	195	0111010110
26	01111111110101011	111	01110010110111100	196	011101101011
27	011111111101010000	112	01110001	197	011100101100000

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

28	01110100101	113	0111010101001	198	0111000010110
29	011111111101010100	114	011101010110110	199	011101001000
30	0111110100010	115	01111010011100	200	011111111111
31	011100001000	116	01110101110011	201	01111110110010110
32	011111100	117	01110110011101101	202	01110110011000000
33	01111010011001	118	011110001100000	203	01110110011111110
34	011111011101011	119	011111011100001	204	011101011101
35	01111101110000001	120	011110111	205	01110110011111101
36	01110100100111	121	01111111110111	206	01111000110010
37	011101010101011001	122	01111000111100	207	0111110100000
38	01110110011111011	123	01110110011100	208	0111101100
39	011110100110001011	124	011110000	209	01110010110100
40	0111010010010	125	01111101110001	210	011110100110000
41	011101010101011010	126	0111100110	211	011110001111010
42	01111010011101101111	127	0111100111	212	01110010110001
43	011101001001100011	128	001	213	01110110011111100
44	0111011001101101	129	01111101111	214	011101010101001
45	01111010011101101110	130	01111111110110	215	011111000000111
46	011101001001100010	131	011100001110	216	011111111110
47	01110010110101001	132	0111100011111	217	0111110111010001
48	011111110	133	01110010110101000	218	011101010101000
49	0111001011001	134	011111000000101	219	0111110111010000
50	011110100111111	135	0111010101011	220	011100001101
51	01110101011010	136	011100001010	221	0111101001110111
52	01111101110010	137	01110110011111111	222	0111010101000
53	01111000111101110	138	011110100110001010	223	0111110111011
54	0111100011000011	139	01111101110100110010	224	000
55	01110010110111110	140	01110110110011	225	0111110101
56	011111001	141	01111010011101101101	226	011101010111

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

57	01111111110100	142	01110110011010	227	0111110001
58	011111011100110	143	0111010111000	228	011111010011
59	0111011001110100	144	01110110100	229	01110000111110
60	011100000	145	011101010110111	230	0111110000000
61	0111100011000010	146	0111011001110111	231	01110110111
62	011110100001	147	01111101110100111	232	01111100001
63	01111010001	148	011111010001100	233	011111111101011
64	01110111	149	011101001001100111	234	011110100111110
65	01111000110011	150	01110010110111011	235	01111000110001
66	0111110100011010	151	011111111101010010	236	01110010111
67	0111011001111001	152	011110100000	237	011101101100000
68	011111011100111	153	01111101000110110	238	0111110100001
69	011101010101011101100001	154	011111011101001101	239	011101101101
70	0111101001110110010	155	011110100110001111	240	010
71	011111111101010001	156	01111110110011	241	0111111110
72	011111101100100	157	011101001001100100	242	011111101101
73	01111101110100110001	158	01110000111111	243	0111111010
74	01111101110100110000	159	01110000101110	244	0111100010
75	011101010101011101100111	160	01111110111	245	0111110100101
76	0111100011110110	161	011110100111010	246	011111000001
77	01111010011101101100	162	011101100110111	247	0111011000
78	0111010101010111011000011	163	0111110111010010	248	1010
79	011101001001100110	164	011111011101010	249	0111101011
80	0111111011000	165	011111101100101110	250	01111010010
81	011101100110001	166	011111111101010101	251	0111101010
82	01111010011000100	167	01110010110111101	252	1011
83	01110010110111111	168	01110110011001	253	011101000
84	0111010101010111	169	011101001001100001	254	0110
85	011101001001100000	170	011110100110001110	255	11

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.3.4 Macroblock pattern

Table B.82 — Variable length codes for coded_block_pattern

coded_block_pattern VLC code	cbp	coded_block_pattern VLC code	cbp
111	60	0001 1100	35
1101	4	0001 1011	13
1100	8	0001 1010	49
1011	16	0001 1001	21
1010	32	0001 1000	41
1001 1	12	0001 0111	14
1001 0	48	0001 0110	50
1000 1	20	0001 0101	22
1000 0	40	0001 0100	42
0111 1	28	0001 0011	15
0111 0	44	0001 0010	51
0110 1	52	0001 0001	23
0110 0	56	0001 0000	43
0101 1	1	0000 1111	25
0101 0	61	0000 1110	37
0100 1	2	0000 1101	26
0100 0	62	0000 1100	38
0011 11	24	0000 1011	29
0011 10	36	0000 1010	45
0011 01	3	0000 1001	53
0011 00	63	0000 1000	57
0010 111	5	0000 0111	30
0010 110	9	0000 0110	46
0010 101	17	0000 0101	54
0010 100	33	0000 0100	58
0010 011	6	0000 0011 1	31
0010 010	10	0000 0011 0	47
0010 001	18	0000 0010 1	55
0010 000	34	0000 0010 0	59
0001 1111	7	0000 0001 1	27
0001 1110	11	0000 0001 0	39
0001 1101	19	0000 0000 1	0 (NOTE)
NOTE — This entry shall not be used with 4:2:0 chrominance structure			

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.83 — VLC table for cbpa in the case of four non-transparent blocks

Code	cbpa
0011	00 00
0010 1	00 01
0010 0	00 10
1001	00 11
0001 1	01 00
0111	01 01
0000 10	01 10
1011	01 11
0001 0	10 00
0000 11	10 01
0101	10 10
1010	10 11
0100	11 00
1000	11 01
0110	11 10
11	11 11

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.84 — VLC table for cbpa in the case of three non transparent blocks

Code	cbpa
011	000
000001	001
00001	010
010	011
00010	100
00011	101
001	110
1	111

Table B.85 — VLC table for cbpa in the case of two non transparent blocks

Code	cbpa
0001	00
001	01
01	10
1	11

Table B.86 — VLC table for cbpa in the case of one non transparent block

Code	cbpa
01	0
1	1

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.3.5 Motion vector

Table B.87 — Variable length codes for motion_code

Variable length code	motion_code[r][s][t]
0000 0011 001	-16
0000 0011 011	-15
0000 0011 101	-14
0000 0011 111	-13
0000 0100 001	-12
0000 0100 011	-11
0000 0100 11	-10
0000 0101 01	-9
0000 0101 11	-8
0000 0111	-7
0000 1001	-6
0000 1011	-5
0000 111	-4
0001 1	-3
0011	-2
011	-1
1	0
010	1
0010	2
0001 0	3
0000 110	4
0000 1010	5
0000 1000	6
0000 0110	7
0000 0101 10	8
0000 0101 00	9
0000 0100 10	10
0000 0100 010	11
0000 0100 000	12
0000 0011 110	13
0000 0011 100	14
0000 0011 010	15
0000 0011 000	16

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.3.6 Sprite Coding

Table B.88 — Code table for the first trajectory point

dmv value	SSS	VLC	dmv_code
-2097151 … -1048576, 1048576 … 2097152	21	111111111111 1111110	00000000000000000000...01111111111111111111 , 10000000000000000000...11111111111111111111
-1048575 … -524288, 524288 … 1048575	20	111111111111 111110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-524287 … -262144, 262144 … -524287	19	111111111111 11110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-262143 … -131072, 131072 …262143	18	111111111111 1110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-131071 … -65536, 65536 … 131071	17	111111111111 110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-65535 … -32768, 32768 … 65535	16	111111111111 10	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-32767 … -16384, 16384 … 32767	15	111111111111 0	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-16383 … -8192, 8192 … 16383	14	111111111110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-8191 … -4096, 4096 … 8191	13	11111111110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-4095 … -2048, 2048 … 4095	12	1111111110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-2047...-1024, 1024...2047	11	111111110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-1023...-512, 512...1023	10	11111110	00000000000000000000...01111111111111111111, 10000000000000000000...11111111111111111111
-511...-256, 256...511	9	1111110	000000000...011111111, 100000000...111111111
-255...-128, 128...255	8	111110	00000000...01111111, 10000000...11111111
-127...-64, 64...127	7	11110	0000000...0111111, 1000000...1111111
-63...-32, 32...63	6	1110	000000...011111, 100000...111111
-31...-16, 16...31	5	110	00000...01111, 10000...1111
-15...-8, 8...15	4	101	0000...0111, 1000...1111
-7...-4, 4...7	3	100	000...011, 100...111
-3...-2, 2...3	2	011	00...01, 10...11
-1, 1	1	010	0, 1
0	0	00	-

Table B.89 — Code table for scaled brightness change factor

brightness_change_ factor value	brightness_ change_factor_ length value	brightness_ change_factor_ length VLC	brightness_change_factor
-16...-1, 1...16	1	0	00000...01111, 10000...11111
-48...-17, 17...48	2	10	000000...011111, 100000...111111
112...-49, 49...112	3	110	0000000...0111111, 1000000...1111111
113…624	4	1110	000000000...111111111
625...1648	4	1111	0000000000…1111111111

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.4 Variable length codes for FGS

VLC Tables for FGS is specified in this Annex.

B.4.1 table_bpc

Table B.90 — VLC table for MSB block-bp if fw_matrix[0] = 0 or 1

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	01
2	1	0	3	101
3	2	0	4	1101
4	3	0	4	0001
5	4	0	5	11110
6	5	0	5	10011
7	6	0	6	110011
8	7	0	6	111010
9	8	0	6	100010
10	9	0	6	100101
11	10	0	7	1110011
12	11	0	7	0000001
13	12	0	8	11101101
14	13	0	8	10001111
15	14	0	9	110000101
16	15	0	9	111011100
17	16	0	9	000000001
18	17	0	9	000010001
19	18	0	9	000010000
20	19	0	10	1100000010
21	20	0	9	110000000
22	21	0	10	1000111010
23	22	0	11	11000011011
24	23	0	11	11101110110
25	24	0	11	10001110010
26	25	0	12	111011001100
27	26	0	12	000000000001
28	27	0	12	100011101100
29	28	0	13	1000111001110
30	29	0	13	1000111001111
31	30	0	14	00000000000001
32	31	0	13	1110010011000
33	32	0	13	1110010011010
34	33	0	13	1110010011011
35	34	0	13	1110010011001
36	35	0	13	0000000000001
37	0	1	3	001
38	1	1	5	11111
39	2	1	5	10000
40	3	1	6	111000
41	4	1	6	000011

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

42	5	1	6	000001
43	6	1	7	1100010
44	7	1	7	1100100
45	8	1	7	1100101
46	9	1	6	100100
47	10	1	7	1100011
48	11	1	7	1000110
49	12	1	7	0000101
50	13	1	8	11000001
51	14	1	8	11100101
52	15	1	8	00000001
53	16	1	9	000010011
54	17	1	9	110000111
55	18	1	9	111001000
56	19	1	9	110000100
57	20	1	8	11101111
58	21	1	9	111011000
59	22	1	9	000010010
60	23	1	10	1100001100
61	24	1	10	1100000011
62	25	1	10	1110111010
63	26	1	11	11000011010
64	27	1	11	11100100100
65	28	1	11	00000000010
66	29	1	11	00000000011
67	30	1	11	10001110111
68	31	1	11	11101100111
69	32	1	11	11100100111
70	33	1	11	11101110111
71	34	1	11	11100100101
72	35	1	10	1000111000
73	36	1	11	00000000001
74	37	1	12	111011001101
75	38	1	12	100011101101
76	39	1	12	100011100110
77	Escape	Code	10	1110110010

Table B.91 — VLC table for MSB-1 block-bp if fw_matrix[0] = 0 or 1

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	01
2	1	0	2	10
3	2	0	3	111
4	3	0	4	0010
5	4	0	4	1100
6	5	0	5	00010
7	6	0	5	11010
8	7	0	6	001110
9	8	0	6	000001

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

10	9	0	7	0011000
11	10	0	7	0001111
12	11	0	7	1101100
13	12	0	8	00011101
14	13	0	8	00000011
15	14	0	9	000110000
16	15	0	10	0011001111
17	16	0	10	0000000011
18	17	0	10	0000001001
19	18	0	11	00110011001
20	19	0	11	00110011011
21	20	0	11	00000000001
22	21	0	12	001100110000
23	22	0	12	000000101001
24	23	0	12	110111110000
25	24	0	13	0011001100011
26	25	0	13	0000001011101
27	26	0	13	1101111100010
28	27	0	15	000000000000001
29	0	1	6	001101
30	1	1	6	000011
31	2	1	7	0011111
32	3	1	7	0001101
33	4	1	7	0000100
34	5	1	7	0000101
35	6	1	7	1101101
36	7	1	8	00110010
37	8	1	8	00111100
38	9	1	8	00111101
39	10	1	8	00011100
40	11	1	8	00011001
41	12	1	8	00000001
42	13	1	8	11011110
43	14	1	9	000110001
44	15	1	10	0011001110
45	16	1	10	0000000010
46	17	1	10	0000000001
47	18	1	10	0000001000
48	19	1	10	1101111101
49	20	1	10	1101111111
50	21	1	11	00110011010
51	22	1	11	00000010110
52	23	1	11	11011111100
53	24	1	11	00000010101
54	25	1	11	11011111001
55	26	1	12	000000000001
56	27	1	12	000000101111
57	28	1	13	0000000000001
58	29	1	13	0000001010001

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

59	30	1	13	0000001011100
60	31	1	13	0000001010000
61	32	1	13	1101111100011
62	33	1	14	00000000000001
63	34	1	14	00110011000100
64	35	1	14	00110011000101
65	All	Zero	7	1101110
66	Escape	Code	11	11011111101

Table B.92 — VLC table for MSB-2 block-bp if fw_matrix[0] = 0 or 1

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	1	1
2	1	0	3	010
3	2	0	3	001
4	3	0	4	0111
5	4	0	5	01100
6	5	0	5	00001
7	6	0	6	011011
8	7	0	6	000101
9	8	0	7	0110100
10	9	0	7	0001110
11	10	0	8	00000101
12	11	0	8	00011001
13	12	0	9	000001000
14	13	0	10	0110101111
15	14	0	10	0000000001
16	15	0	11	00000100101
17	16	0	12	011010111011
18	17	0	12	000001001000
19	18	0	13	0110101110001
20	19	0	13	0110101101100
21	20	0	13	0110101101110
22	21	0	13	0001111000110
23	22	0	14	01101011100000
24	23	0	15	000000000000001
25	0	1	6	000100
26	1	1	7	0000001
27	2	1	7	0001101
28	3	1	8	01101010
29	4	1	8	00000110
30	5	1	8	00000001
31	6	1	8	00011111
32	7	1	8	00011000
33	8	1	9	000001111
34	9	1	9	000001110
35	10	1	9	000000001
36	11	1	9	000111101
37	12	1	10	0110101100

--`,,`-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

38	13	1	10	0000010011
39	14	1	10	0001111001
40	15	1	11	00000000001
41	16	1	12	011010111010
42	17	1	12	011010111001
43	18	1	12	000001001001
44	19	1	12	000000000001
45	20	1	12	000111100010
46	21	1	13	0110101101101
47	22	1	13	0110101101111
48	23	1	13	0000000000001
49	24	1	13	0001111000111
50	25	1	14	01101011100001
51	26	1	14	00000000000001
52	All	Zero	11	01101011010
53	Escape	Code	11	00011110000

Table B.93 — VLC table for MSB-3 block-bp and other block-bps if fw_matrix[0] = 0 or 1

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	1	1
2	1	0	2	01
3	2	0	3	001
4	3	0	5	00010
5	4	0	5	00001
6	5	0	6	000001
7	6	0	7	0001100
8	7	0	8	00011101
9	8	0	9	000111001
10	9	0	9	000000011
11	10	0	10	0000000010
12	11	0	11	00011010100
13	12	0	11	00000001010
14	13	0	12	000000000001
15	14	0	13	0000000011110
16	15	0	13	0000000101101
17	16	0	14	00000000000011
18	0	1	7	0001111
19	1	1	7	0000001
20	2	1	8	00011011
21	3	1	9	000111000
22	4	1	9	000110100
23	5	1	10	0001101011
24	6	1	10	0000000001
25	7	1	10	0000000100
26	8	1	11	00000000110
27	9	1	11	00000000001
28	10	1	12	000110101011
29	11	1	12	000000001110

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

30	12	1	12	000000010111
31	13	1	13	0000000011111
32	14	1	13	0000000101100
33	15	1	14	00000000000001
34	16	1	14	00000000000010
35	17	1	15	000000000000001
36	All	Zero	16	0000000000000001
37	Escape	Code	12	000110101010

The above VLC tables are implemented as the following arrays in C. The small array table_symbol[4][3] indicates the number of entries in the VLC tables for EOP=0, EOP=1, and ALL_ZERO. For example, the first row {36, 40, 0} indicates that the VLC table for the MSB block-bp has 36 RUN entries for EOP=0, 40 RUN entries for EOP=1, and no ALL_ZERO entry before the Escape Code. The second row {28, 36, 1} means that the VLC table for the MSB-1 block-bp has 28 RUN entries for EOP=0, 36 RUN entries for EOP=1, and 1 ALL_ZERO entry before the Escape Code. The large array table_bpc[4][154] contains the 4 VLC tables. The array contains all the VLC Codes and Code Lengths in the format of the decimal value of a VLC Code and its Code Length as a pair. For example, the first pair of numbers in table_bpc[4][154] is (1,2) that corresponds to the first VLC Code in the VLC Table for the MSB Block-bp, 01, with a Code Length 2. The second pair (5,3) corresponds to the second VLC Code, 101, in the same table with a Code Length 3.

```
int table_symbol[4][3] = {

{ 36,  40,  0},

{ 28,  36,  1},

{ 24,  27,  1},

{ 17,  18,  1},

};

int table_bpc[4][154] = {

{

    1, 2, 5, 3, 13, 4, 1, 4, 30, 5, 19, 5, 51, 6, 58, 6, 34, 6, 37, 6, 115, 7, 1, 7,  237, 8, 143, 8, 389, 9, 476, 9, 1, 9, 17, 9, 16, 9, 770,10, 384, 9, 570,10, 1563,11, 1910,11, 1138,11, 3788,12, 1,12, 2284,12, 4558,13, 4559,13, 1,14, 7320,13, 7322,13, 7323,13, 7321,13, 1,13, 1, 3, 31, 5, 16, 5, 56, 6, 3, 6, 1, 6, 98, 7, 100, 7, 101, 7, 36, 6, 99, 7, 70, 7, 5, 7, 193, 8, 229, 8, 1, 8, 19, 9, 391, 9, 456, 9, 388, 9, 239, 8, 472, 9, 18, 9, 780,10, 771,10, 954,10, 1562,11, 1828,11, 2,11, 3,11, 1143,11, 1895,11, 1831,11, 1911,11, 1829,11, 568,10, 1,11, 3789,12, 2285,12, 2278,12, 946,10},

{

    1, 2, 2, 2, 7, 3, 2, 4, 12, 4, 2, 5, 26, 5, 14, 6, 1, 6, 24, 7, 15, 7, 108, 7, 29, 8, 3, 8, 48, 9, 207,10, 3,10, 9,10, 409,11, 411,11, 1,11, 816,12, 41,12, 3568,12, 1635,13, 93,13, 7138,13, 1,15, 13, 6, 3, 6, 31, 7, 13, 7, 4, 7, 5, 7, 109, 7, 50, 8, 60, 8, 61, 8, 28, 8, 25, 8, 1, 8, 222, 8, 49, 9, 206,10, 2,10, 1,10, 8,10, 893,10, 895,10, 410,11, 22,11, 1788,11, 21,11, 1785,11, 1,12, 47,12, 1,13, 81,13, 92,13, 80,13, 7139,13, 1,14, 3268,14, 3269,14, 110, 7, 1789,11},

{

    1, 1, 2, 3, 1, 3, 7, 4, 12, 5, 1, 5, 27, 6, 5, 6, 52, 7, 14, 7, 5, 8, 25, 8, 8, 9, 431,10, 1,10, 37,11, 1723,12, 72,12, 3441,13, 3436,13, 3438,13, 966,13, 6880,14, 1,15, 4, 6, 1, 7, 13, 7, 106, 8, 6, 8, 1, 8, 31, 8, 24, 8, 15, 9, 14, 9, 1, 9, 61, 9, 428,10, 19,10, 121,10, 1,11, 1722,12, 1721,12, 73,12, 1,12, 482,12, 3437,13, 3439,13, 1,13, 967,13, 6881,14, 1,14, 858,11, 240,11},
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

{

1, 1, 1, 2, 1, 3, 2, 5, 1, 5, 1, 6, 12, 7, 29, 8, 57, 9, 3, 9, 2,10, 212,11, 10,11, 1,12, 30,13, 45,13, 3,14, 15, 7, 1, 7, 27, 8, 56, 9, 52, 9, 107,10, 1,10, 4,10, 6,11, 1,11, 427,12, 14,12, 23,12, 31,13, 44,13, 1,14, 2,14, 1,15, 1,16, 426,12}

};

B.4.2 table_bpc4fw_m

Table B.94 — VLC table for MSB block-bp if fw_matrix[0] = 2 or 3

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	10
2	1	0	4	0110
3	2	0	5	01110
4	3	0	6	011110
5	4	0	6	010111
6	5	0	6	000011
7	6	0	6	000110
8	7	0	7	0101010
9	8	0	7	0101000
10	9	0	7	0000001
11	10	0	8	01111111
12	11	0	8	01011001
13	12	0	8	00010111
14	13	0	9	011111000
15	14	0	9	010101101
16	15	0	9	000000001
17	16	0	10	0111110011
18	17	0	10	0111110010
19	18	0	10	0101011110
20	19	0	10	0101101101
21	20	0	11	00000000001
22	0	1	2	11
23	1	1	3	001
24	2	1	4	0100
25	3	1	6	000100
26	4	1	7	0101001
27	5	1	7	0000010
28	6	1	7	0001010
29	7	1	7	0001110
30	8	1	7	0001111
31	9	1	8	01111101
32	10	1	8	01111110
33	11	1	8	01011000
34	12	1	8	00000001
35	13	1	8	00000110
36	14	1	8	00010110
37	15	1	9	010101100
38	16	1	9	010110111

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

39	17	1	9	010101110
40	18	1	9	010110101
41	19	1	9	010110100
42	20	1	9	000001110
43	21	1	10	0101011111
44	22	1	10	0000011111
45	23	1	10	0000000001
46	24	1	10	0000011110
47	25	1	10	0101101100
48	Escape	Code	6	000010

Table B.95 — VLC table for MSB-1 block-bp if fw_matrix[0] = 2 or 3

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	01
2	1	0	3	001
3	2	0	3	100
4	3	0	4	1101
5	4	0	5	00001
6	5	0	5	11100
7	6	0	6	000110
8	7	0	6	000001
9	8	0	6	110001
10	9	0	7	0001111
11	10	0	7	1110111
12	11	0	7	1010010
13	12	0	8	11101001
14	13	0	8	11000010
15	14	0	9	000111000
16	15	0	9	110000111
17	16	0	9	101010100
18	17	0	10	0000000011
19	18	0	10	1110110111
20	19	0	10	1100001101
21	20	0	10	1010111010
22	21	0	11	00000000001
23	22	0	11	11000011001
24	23	0	11	10101010111
25	24	0	12	000111001010
26	0	1	4	1111
27	1	1	5	00010
28	2	1	5	11001
29	3	1	6	101000
30	4	1	7	0000001
31	5	1	7	1110101
32	6	1	7	1010100
33	7	1	7	1010110
34	8	1	7	1010011
35	9	1	8	00011101

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

36	10	1	8	00000001
37	11	1	8	11101000
38	12	1	8	11101100
39	13	1	8	11000000
40	14	1	8	10101111
41	15	1	9	111011010
42	16	1	9	101010111
43	17	1	9	101010110
44	18	1	9	101011100
45	19	1	10	0000000010
46	20	1	10	0000000001
47	21	1	10	1110110110
48	22	1	10	1010101010
49	23	1	10	1010111011
50	24	1	11	00011100100
51	25	1	11	00011100110
52	26	1	11	11000011000
53	27	1	11	10101010110
54	28	1	12	000111001011
55	29	1	12	000111001111
56	30	1	12	000111001110
57	31	1	12	000000000001
58	32	1	13	0000000000001
59	All	Zero	4	1011
60	Escape	Code	8	11000001

Table B.96 — VLC table for MSB-2 block-bp if fw_matrix[0] = 2 or 3

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	01
2	1	0	2	10
3	2	0	3	110
4	3	0	4	0001
5	4	0	4	1110
6	5	0	5	00101
7	6	0	5	11110
8	7	0	6	000001
9	8	0	6	111111
10	9	0	7	0000001
11	10	0	7	0011010
12	11	0	8	00001100
13	12	0	8	00110010
14	13	0	9	000010010
15	14	0	9	001101101
16	15	0	10	0000100111
17	16	0	10	0000000001
18	17	0	11	00001011111
19	18	0	11	00000000100
20	19	0	11	00000000001

--`,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

21	20	0	11	00110110000
22	21	0	12	000011111011
23	22	0	12	001001010010
24	23	0	13	0000101111010
25	24	0	13	0000100110010
26	25	0	14	00000000000001
27	0	1	5	00111
28	1	1	6	001000
29	2	1	6	111110
30	3	1	7	0010011
31	4	1	7	0011000
32	5	1	8	00001010
33	6	1	8	00001110
34	7	1	8	00001101
35	8	1	8	00000001
36	9	1	8	00100100
37	10	1	8	00110011
38	11	1	8	00110111
39	12	1	9	000010110
40	13	1	9	000011110
41	14	1	9	001001011
42	15	1	10	0000111111
43	16	1	10	0000000011
44	17	1	10	0010010101
45	18	1	10	0011011001
46	19	1	11	00001001101
47	20	1	11	00001111100
48	21	1	11	00000000101
49	22	1	11	00100101000
50	23	1	11	00110110001
51	24	1	12	000010111100
52	25	1	12	000010011000
53	26	1	12	000000000001
54	27	1	12	001001010011
55	28	1	13	0000101111011
56	29	1	13	0000100110011
57	30	1	13	0000111110101
58	31	1	13	0000111110100
59	32	1	13	0000000000001
60	All	Zero	8	00001000
61	Escape	Code	10	0000101110

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.97 — VLC table for MSB-3 block-bp and other block-bps if fw_matrix[0] = 2 or 3

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	1	1
2	1	0	3	001
3	2	0	3	010
4	3	0	4	0001
5	4	0	5	00001
6	5	0	5	01111
7	6	0	6	000001
8	7	0	6	011100
9	8	0	7	0110100
10	9	0	7	0111010
11	10	0	8	01101111
12	11	0	8	01110111
13	12	0	9	011011101
14	13	0	10	0000001001
15	14	0	10	0110110010
16	15	0	11	00000000001
17	16	0	11	01100110110
18	17	0	12	000000100001
19	18	0	12	011011001100
20	19	0	12	011001110010
21	20	0	13	0000001000100
22	21	0	13	0110110011110
23	22	0	14	00000010001010
24	23	0	14	00000000000001
25	0	1	6	011000
26	1	1	7	0110101
27	2	1	7	0110010
28	3	1	8	00000011
29	4	1	8	00000001
30	5	1	8	01101101
31	6	1	8	01110110
32	7	1	9	000000101
33	8	1	9	011011100
34	9	1	9	011011000
35	10	1	9	011001100
36	11	1	9	011001111
37	12	1	10	0000000001
38	13	1	10	0000000011
39	14	1	10	0110011010
40	15	1	11	00000000101
41	16	1	11	01100110111
42	17	1	11	01100111000
43	18	1	12	000000100011
44	19	1	12	000000000001
45	20	1	12	011011001110
46	21	1	12	011011001101
47	22	1	12	011001110011

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48	23	1	13	0000001000001
49	24	1	13	0000000000001
50	25	1	13	0110110011111
51	26	1	14	00000010001011
52	27	1	14	00000010000001
53	28	1	14	00000010000000
54	All	Zero	11	00000000100
55	Escape	Code	10	0110011101

int table_symbol4fw_m[4][3] = {

{ 21, 26, 0},

{ 25, 33, 1},

{ 26, 33, 1},

{ 24, 29, 1},

};

int table_bpc4fw_m[4][122] = {

{

2, 2, 6, 4, 14, 5, 30, 6, 23, 6, 3, 6, 6, 6, 42, 7, 40, 7, 1, 7, 127, 8, 89, 8, 23, 8, 248, 9, 173, 9, 1, 9, 499,10, 498,10, 350,10, 365,10, 1,11, 3, 2, 1, 3, 4, 4, 4, 6, 41, 7, 2, 7, 10, 7, 14, 7, 15, 7, 125, 8, 126, 8, 88, 8, 1, 8, 6, 8, 22, 8, 172, 9, 183, 9, 174, 9, 181, 9, 180, 9, 14, 9, 351,10, 31,10, 1,10, 30,10, 364,10, 2, 6},

{

1, 2, 1, 3, 4, 3, 13, 4, 1, 5, 28, 5, 6, 6, 1, 6, 49, 6, 15, 7, 119, 7, 82, 7, 233, 8, 194, 8, 56, 9, 391, 9, 340, 9, 3,10, 951,10, 781,10, 698,10, 1,11, 1561,11, 1367,11, 458,12, 15, 4, 2, 5, 25, 5, 40, 6, 1, 7, 117, 7, 84, 7, 86, 7, 83, 7, 29, 8, 1, 8, 232, 8, 236, 8, 192, 8, 175, 8, 474, 9, 343, 9, 342, 9, 348, 9, 2,10, 1,10, 950,10, 682,10, 699,10, 228,11, 230,11, 1560,11, 1366,11, 459,12, 463,12, 462,12, 1,12, 1,13, 11, 4, 193, 8},

{

1, 2, 2, 2, 6, 3, 1, 4, 14, 4, 5, 5, 30, 5, 1, 6, 63, 6, 1, 7, 26, 7, 12, 8, 50, 8, 18, 9, 109, 9, 39,10, 1,10, 95,11, 4,11, 1,11, 432,11, 251,12, 594,12, 378,13, 306,13, 1,14, 7, 5, 8, 6, 62, 6, 19, 7, 24, 7, 10, 8, 14, 8, 13, 8, 1, 8, 36, 8, 51, 8, 55, 8, 22, 9, 30, 9, 75, 9, 63,10, 3,10, 149,10, 217,10, 77,11, 124,11, 5,11, 296,11, 433,11, 188,12, 152,12, 1,12, 595,12, 379,13, 307,13, 501,13, 500,13, 1,13, 8, 8, 46,10},

{

1, 1, 1, 3, 2, 3, 1, 4, 1, 5, 15, 5, 1, 6, 28, 6, 52, 7, 58, 7, 111, 8, 119, 8, 221, 9, 9,10, 434,10, 1,11, 822,11, 33,12, 1740,12, 1650,12, 68,13, 3486,13, 138,14, 1,14, 24, 6, 53, 7, 50, 7, 3, 8, 1, 8, 109, 8, 118, 8, 5, 9, 220, 9, 216, 9, 204, 9, 207, 9, 1,10, 3,10, 410,10, 5,11, 823,11, 824,11, 35,12, 1,12, 1742,12, 1741,12, 1651,12, 65,13, 1,13, 3487,13, 139,14, 129,14, 128,14, 4,11, 413,10}

};

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.4.3 table_bpc4fw_h

Table B.98 — VLC table for MSB block-bp if fw_matrix[0] = 4, 5, 6, or 7

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	01
2	1	0	5	00001
3	2	0	7	0000011
4	3	0	9	000000001
5	0	1	1	1
6	1	1	3	001
7	2	1	4	0001
8	3	1	7	0000010
9	4	1	8	00000011
10	5	1	8	00000001
11	Escape	Code	8	00000010

Table B.99 — VLC table for MSB-1 block-bp if fw_matrix[0] = 4, 5, 6, or 7

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	10
2	1	0	3	011
3	2	0	4	0001
4	3	0	6	110001
5	4	0	6	000010
6	5	0	8	00000010
7	6	0	9	000001000
8	7	0	9	000000011
9	8	0	9	000000010
10	9	0	10	0000000010
11	10	0	11	00000000001
12	0	1	3	111
13	1	1	3	001
14	2	1	4	1101
15	3	1	5	11001
16	4	1	6	110000
17	5	1	6	000011
18	6	1	8	00000101
19	7	1	9	000001001
20	8	1	9	000000111
21	9	1	9	000000110
22	10	1	10	0000000011
23	11	1	10	0000000001
24	All	Zero	3	010
25	Escape	Code	7	0000011

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B.100 — VLC table for MSB-2 block-bp if fw_matrix[0] = 4, 5, 6, or 7

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	10
2	1	0	2	01
3	2	0	4	1100
4	3	0	5	11010
5	4	0	5	00001
6	5	0	6	000001
7	6	0	7	1111101
8	7	0	8	11110011
9	8	0	8	00011110
10	9	0	8	00000001
11	10	0	9	111110000
12	11	0	9	000000100
13	12	0	10	1111000000
14	13	0	10	1111100101
15	14	0	10	0000001010
16	15	0	11	11110000110
17	16	0	11	11111001000
18	0	1	3	001
19	1	1	4	1110
20	2	1	5	11011
21	3	1	5	00010
22	4	1	6	111111
23	5	1	6	000110
24	6	1	7	0001110
25	7	1	8	11110001
26	8	1	8	00011111
27	9	1	8	00000011
28	10	1	9	111110001
29	11	1	9	111110011
30	12	1	9	000000001
31	13	1	10	1111000001
32	14	1	10	1111000010
33	15	1	10	0000000001
34	16	1	11	11110000111
35	17	1	11	11111001001
36	18	1	11	00000010111
37	19	1	11	00000010110
38	20	1	11	00000000001
39	21	1	12	000000000001
40	All	Zero	6	111101
41	Escape	Code	8	11110010

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table B.101 — VLC table for MSB-3 block-bp and other block-bps if fw_matrix[0] = 4, 5, 6, or 7

Entry Number	RUN	EOP	Code Length	VLC Code
1	0	0	2	01
2	1	0	2	10
3	2	0	3	110
4	3	0	4	0001
5	4	0	5	00111
6	5	0	6	001100
7	6	0	6	001001
8	7	0	6	111011
9	8	0	7	0000011
10	9	0	8	00110111
11	10	0	8	00000011
12	11	0	8	11101011
13	12	0	9	000000001
14	13	0	9	111010010
15	14	0	10	0000000001
16	15	0	10	1110100010
17	16	0	11	00000000001
18	17	0	11	11101001111
19	18	0	12	000000011011
20	19	0	12	000000000001
21	20	0	12	000011000011
22	21	0	12	111010001101
23	22	0	13	0000000000001
24	0	1	4	1111
25	1	1	5	00101
26	2	1	5	11100
27	3	1	6	001000
28	4	1	6	000010
29	5	1	7	0011010
30	6	1	7	0000111
31	7	1	8	00110110
32	8	1	8	00000010
33	9	1	8	00001101
34	10	1	8	11101010
35	11	1	9	000000010
36	12	1	9	000001011
37	13	1	9	000011001
38	14	1	9	111010000
39	15	1	10	1110100110
40	16	1	11	00000001100
41	17	1	11	00000001111
42	18	1	11	00001100000
43	19	1	11	00001100010
44	20	1	11	11101001110
45	21	1	11	00001100011
46	22	1	11	11101000111
47	23	1	12	000000011101

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

48	24	1	12	000000011010
49	25	1	12	000000011100
50	26	1	12	000011000010
51	27	1	12	111010001100
52	28	1	14	00000000000001
53	All	Zero	8	00000100
54	Escape	Code	9	000001010

int table_symbol4fw_h[4][3] = {

{ 4, 6, 0},

{ 11, 12, 1},

{ 17, 22, 1},

{ 23, 29, 1},

};

int table_bpc4fw_h[4][108] = {

{

1, 2, 1, 5, 3, 7, 1, 9, 1, 1, 1, 3, 1, 4, 2, 7, 3, 8, 1, 8, 2, 8},

{

2, 2, 3, 3, 1, 4, 49, 6, 2, 6, 2, 8, 8, 9, 3, 9, 2, 9, 2,10, 1,11, 7, 3, 1, 3, 13, 4, 25, 5, 48, 6, 3, 6, 5, 8, 9, 9, 7, 9, 6, 9, 3,10, 1,10, 2, 3, 3, 7},

{

2, 2, 1, 2, 12, 4, 26, 5, 1, 5, 1, 6, 125, 7, 243, 8, 30, 8, 1, 8, 496, 9, 4, 9, 960,10, 997,10, 10,10, 1926,11, 1992,11, 1, 3, 14, 4, 27, 5, 2, 5, 63, 6, 6, 6, 14, 7, 241, 8, 31, 8, 3, 8, 497, 9, 499, 9, 1, 9, 961,10, 962,10, 1,10, 1927,11, 1993,11, 23,11, 22,11, 1,11, 1,12, 61, 6, 242, 8},

{

1, 2, 2, 2, 6, 3, 1, 4, 7, 5, 12, 6, 9, 6, 59, 6, 3, 7, 55, 8, 3, 8, 235, 8, 1, 9, 466, 9, 1,10, 930,10, 1,11, 1871,11, 27,12, 1,12, 195,12, 3725,12, 1,13, 15, 4, 5, 5, 28, 5, 8, 6, 2, 6, 26, 7, 7, 7, 54, 8, 2, 8, 13, 8, 234, 8, 2, 9, 11, 9, 25, 9, 464, 9, 934,10, 12,11, 15,11, 96,11, 98,11, 1870,11, 99,11, 1863,11, 29,12, 26,12, 28,12, 194,12, 3724,12, 1,14, 4, 8, 10, 9}

};

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C
(normative)

Face and body object decoding tables and definitions

FAPs names may contain letters with the following meaning: l = left, r = right, t = top, b = bottom, i = inner, o = outer, m = middle. The sum of two corresponding top and bottom eyelid FAPs must equal 1024 when the eyelids are closed. Inner lips are closed when the sum of two corresponding top and bottom lip FAPs equals zero. For example: (lower_t_midlip + raise_b_midlip) = 0 when the lips are closed. All directions are defined with respect to the face and not the image of the face.

Table C.1 — FAP definitions, group assignments and step sizes

#	FAP name	FAP description	units	Uni-orBi dir	Pos motion	Grp	FDP sub-grp num	Quan-tisa-tion step size	Min/Max I-Frame quantised values	Min/Max P-Frame quantised values
1	viseme	Set of values determining the mixture of two visemes for this frame (e.g. pbm, fv, th)	na	na	na	1	na	1	viseme_ blend: +63	viseme_ blend: +-63
2	expression	A set of values determining the mixture of two facial expression	na	na	na	1	na	1	expression _intensity1, expression _intensity2: +63	expression _intensity1, expression _intensity2: +-63
3	open_jaw	Vertical jaw displacement (does not affect mouth opening)	MNS	U	down	2	1	4	+1080	+-360
4	lower_t_midlip	Vertical top middle inner lip displacement	MNS	B	down	2	2	2	+-600	+-180
5	raise_b_midlip	Vertical bottom middle inner lip displacement	MNS	B	up	2	3	2	+-1860	+-600
6	stretch_l_cornerlip	Horizontal displacement of left inner lip corner	MW	B	left	2	4	2	+-600	+-180
7	stretch_r_cornerlip	Horizontal displacement of right inner lip corner	MW	B	right	2	5	2	+-600	+-180
8	lower_t_lip_lm	Vertical displacement of midpoint between left corner and middle of top inner lip	MNS	B	down	2	6	2	+-600	+-180
9	lower_t_lip_rm	Vertical displacement of midpoint between	MNS	B	down	2	7	2	+-600	+-180

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

		right corner and middle of top inner lip								
10	raise_b_lip_lm	Vertical displacement of midpoint between left corner and middle of bottom inner lip	MNS	B	up	2	8	2	+-1860	+-600
11	raise_b_lip_rm	Vertical displacement of midpoint between right corner and middle of bottom inner lip	MNS	B	up	2	9	2	+-1860	+-600
12	raise_l_cornerlip	Vertical displacement of left inner lip corner	MNS	B	up	2	4	2	+-600	+-180
13	raise_r_cornerlip	Vertical displacement of right inner lip corner	MNS	B	up	2	5	2	+-600	+-180
14	thrust_jaw	Depth displacement of jaw	MNS	U	forward	2	1	1	+600	+-180
15	shift_jaw	Side to side displacement of jaw	MW	B	right	2	1	1	+-1080	+-360
16	push_b_lip	Depth displacement of bottom middle lip	MNS	B	forward	2	3	1	+-1080	+-360
17	push_t_lip	Depth displacement of top middle lip	MNS	B	forward	2	2	1	+-1080	+-360
18	depress_chin	Upward and compressing movement of the chin (like in sadness)	MNS	B	up	2	10	1	+-420	+-180
19	close_t_l_eyelid	Vertical displacement of top left eyelid	IRISD	B	down	3	1	1	+-1080	+-600
20	close_t_r_eyelid	Vertical displacement of top right eyelid	IRISD	B	down	3	2	1	+-1080	+-600
21	close_b_l_eyelid	Vertical displacement of bottom left eyelid	IRISD	B	up	3	3	1	+-600	+-240
22	close_b_r_eyelid	Vertical displacement of bottom right eyelid	IRISD	B	up	3	4	1	+-600	+-240
23	yaw_l_eyeball	Horizontal orientation of left eyeball	AU	B	left	3	na	128	+-1200	+-420

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

24	yaw_r_eyeball	Horizontal orientation of right eyeball	AU	B	left	3	na	128	+-1200	+-420
25	pitch_l_eyeball	Vertical orientation of left eyeball	AU	B	down	3	na	128	+-900	+-300
26	pitch_r_eyeball	Vertical orientation of right eyeball	AU	B	down	3	na	128	+-900	+-300
27	thrust_l_eyeball	Depth displacement of left eyeball	ES	B	forward	3	na	1	+-600	+-180
28	thrust_r_eyeball	Depth displacement of right eyeball	ES	B	forward	3	na	1	+-600	+-180
29	dilate_l_pupil	Dilation of left pupil	IRISD	B	growing	3	5	1	+-420	+-120
30	dilate_r_pupil	Dilation of right pupil	IRISD	B	growing	3	6	1	+-420	+-120
31	raise_l_i_eyebrow	Vertical displacement of left inner eyebrow	ENS	B	up	4	1	2	+-900	+-360
32	raise_r_i_eyebrow	Vertical displacement of right inner eyebrow	ENS	B	up	4	2	2	+-900	+-360
33	raise_l_m_eyebrow	Vertical displacement of left middle eyebrow	ENS	B	up	4	3	2	+-900	+-360
34	raise_r_m_eyebrow	Vertical displacement of right middle eyebrow	ENS	B	up	4	4	2	+-900	+-360
35	raise_l_o_eyebrow	Vertical displacement of left outer eyebrow	ENS	B	up	4	5	2	+-900	+-360
36	raise_r_o_eyebrow	Vertical displacement of right outer eyebrow	ENS	B	up	4	6	2	+-900	+-360
37	squeeze_l_eyebrow	Horizontal displacement of left eyebrow	ES	B	right	4	1	1	+-900	+-300
38	squeeze_r_eyebrow	Horizontal displacement of right eyebrow	ES	B	left	4	2	1	+-900	+-300
39	puff_l_cheek	Horizontal displacement of left cheeck	ES	B	left	5	1	2	+-900	+-300
40	puff_r_cheek	Horizontal displacement of right cheeck	ES	B	right	5	2	2	+-900	+-300
41	lift_l_cheek	Vertical displacement of left cheek	ENS	U	up	5	3	2	+-600	+-180
42	lift_r_cheek	Vertical displacement of right cheek	ENS	U	up	5	4	2	+-600	+-180

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

43	shift_tongue_tip	Horizontal displacement of tongue tip	MW	B	right	6	1	1	+-1080	+-420
44	raise_tongue_tip	Vertical displacement of tongue tip	MNS	B	up	6	1	1	+-1080	+-420
45	thrust_tongue_tip	Depth displacement of tongue tip	MW	B	forward	6	1	1	+-1080	+-420
46	raise_tongue	Vertical displacement of tongue	MNS	B	up	6	2	1	+-1080	+-420
47	tongue_roll	Rolling of the tongue into U shape	AU	U	concave upward	6	3, 4	512	+300	+-60
48	head_pitch	Head pitch angle from top of spine	AU	B	down	7	na	170	+-1860	+-600
49	head_yaw	Head yaw angle from top of spine	AU	B	left	7	na	170	+-1860	+-600
50	head_roll	Head roll angle from top of spine	AU	B	right	7	na	170	+-1860	+-600
51	lower_t_midlip_o	Vertical top middle outer lip displacement	MNS	B	down	8	1	2	+-600	+-180
52	raise_b_midlip_o	Vertical bottom middle outer lip displacement	MNS	B	up	8	2	2	+-1860	+-600
53	stretch_l_cornerlip_o	Horizontal displacement of left outer lip corner	MW	B	left	8	3	2	+-600	+-180
54	stretch_r_cornerlip_o	Horizontal displacement of right outer lip corner	MW	B	right	8	4	2	+-600	+-180
55	lower_t_lip_lm_o	Vertical displacement of midpoint between left corner and middle of top outer lip	MNS	B	down	8	5	2	+-600	+-180
56	lower_t_lip_rm_o	Vertical displacement of midpoint between right corner and middle of top outer lip	MNS	B	down	8	6	2	+-600	+-180
57	raise_b_lip_lm_o	Vertical displacement of midpoint between left corner and middle of bottom outer lip	MNS	B	up	8	7	2	+-1860	+-600

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

58	raise_b_lip_rm_o	Vertical displacement of midpoint between right corner and middle of bottom outer lip	MNS	B	up	8	8	2	+-1860	+-600
59	raise_l_cornerlip_o	Vertical displacement of left outer lip corner	MNS	B	up	8	3	2	+-600	+-180
60	raise_r_cornerli _o	Vertical displacement of right outer lip corner	MNS	B	up	8	4	2	+-600	+-180
61	stretch_l_nose	Horizontal displacement of left side of nose	ENS	B	left	9	1	1	+-540	+-120
62	stretch_r_nose	Horizontal displacement of right side of nose	ENS	B	right	9	2	1	+-540	+-120
63	raise_nose	Vertical displacement of nose tip	ENS	B	up	9	3	1	+-680	+-180
64	bend_nose	Horizontal displacement of nose tip	ENS	B	right	9	3	1	+-900	+-180
65	raise_l_ear	Vertical displacement of left ear	ENS	B	up	10	1	1	+-900	+-240
66	raise_r_ear	Vertical displacement of right ear	ENS	B	up	10	2	1	+-900	+-240
67	pull_l_ear	Horizontal displacement of left ear	ENS	B	left	10	3	1	+-900	+-300
68	pull_r_ear	Horizontal displacement of right ear	ENS	B	right	10	4	1	+-900	+-300

Table C.2 — FAP grouping

Group	Number of FAPs
1: visemes and expressions	2
2: jaw, chin, inner lowerlip, cornerlips, midlip	16
3: eyeballs, pupils, eyelids	12
4: eyebrow	8
5: cheeks	4
6: tongue	5
7: head rotation	3
8: outer lip positions	10
9: nose	4
10: ears	4

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the following, each facial expression is defined by a textual description and a pictorial example. This reference was also used for the characteristics of the described expressions.

Table C.3 — Values for expression_select

expression_select	expression name	textual description
0	na	na
1	joy	The eyebrows are relaxed. The mouth is open and the mouth corners pulled back toward the ears.
2	sadness	The inner eyebrows are bent upward. The eyes are slightly closed. The mouth is relaxed.
3	anger	The inner eyebrows are pulled downward and together. The eyes are wide open. The lips are pressed against each other or opened to expose the teeth.
4	fear	The eyebrows are raised and pulled together. The inner eyebrows are bent upward. The eyes are tense and alert.
5	disgust	The eyebrows and eyelids are relaxed. The upper lip is raised and curled, often asymmetrically.
6	surprise	The eyebrows are raised. The upper eyelids are wide open, the lower relaxed. The jaw is opened.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

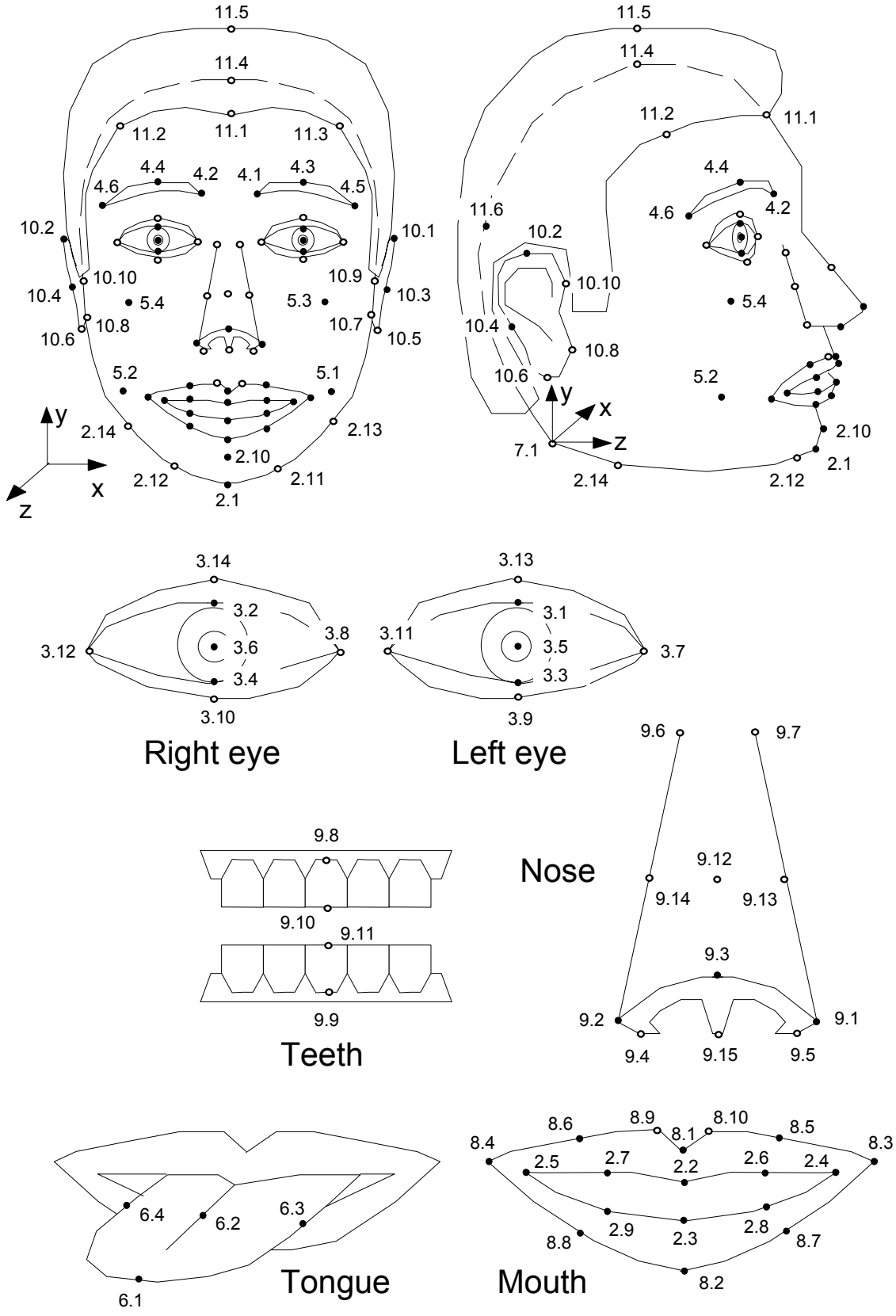


Figure C.1 — FDP feature point set

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the following, the notation 2.1.x indicates the x coordinate of feature point 2.1.

Feature points		**Recommended location constraints**		
#	Text description	x	y	z
2.1	Bottom of the chin	7.1.x		
2.2	Middle point of inner upper lip contour	7.1.x		
2.3	Middle point of inner lower lip contour	7.1.x		
2.4	Left corner of inner lip contour			
2.5	Right corner of inner lip contour			
2.6	Midpoint between f.p. 2.2 and 2.4 in the inner upper lip contour	(2.2.x+2.4.x)/2		
2.7	Midpoint between f.p. 2.2 and 2.5 in the inner upper lip contour	(2.2.x+2.5.x)/2		
2.8	Midpoint between f.p. 2.3 and 2.4 in the inner lower lip contour	(2.3.x+2.4.x)/2		
2.9	Midpoint between f.p. 2.3 and 2.5 in the inner lower lip contour	(2.3.x+2.5.x)/2		
2.10	Chin boss	7.1.x		
2.11	Chin left corner	> 8.7.x and < 8.3.x		
2.12	Chin right corner	> 8.4.x and < 8.8.x		
2.13	Left corner of jaw bone			
2.14	Right corner of jaw bone			
3.1	Center of upper inner left eyelid	(3.7.x+3.11.x)/2		
3.2	Center of upper inner right eyelid	(3.8.x+3.12.x)/2		
3.3	Center of lower inner left eyelid	(3.7.x+3.11.x)/2		
3.4	Center of lower inner right eyelid	(3.8.x+3.12.x)/2		
3.5	Center of the pupil of left eye			
3.6	Center of the pupil of right eye			
3.7	Left corner of left eye			
3.8	Left corner of right eye			
3.9	Center of lower outer left eyelid	(3.7.x+3.11.x)/2		
3.10	Center of lower outer right eyelid	(3.7.x+3.11.x)/2		
3.11	Right corner of left eye			
3.12	Right corner of right eye			
3.13	Center of upper outer left eyelid	(3.8.x+3.12.x)/2		
3.14	Center of upper outer right eyelid	(3.8.x+3.12.x)/2		
4.1	Right corner of left eyebrow			
4.2	Left corner of right eyebrow			
4.3	Uppermost point of the left eyebrow	(4.1.x+4.5.x)/2 or x coord of the uppermost point of the contour		
4.4	Uppermost point of the right eyebrow	(4.2.x+4.6.x)/2 or x coord of the uppermost point of the contour		
4.5	Left corner of left eyebrow			
4.6	Right corner of right eyebrow			

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

5.1	Center of the left cheek		8.3.y	
5.2	Center of the right cheek		8.4.y	
5.3	Left cheek bone	> 3.5.x and < 3.7.x	> 9.15.y and < 9.12.y	
5.4	Right cheek bone	> 3.6.x and < 3.12.x	> 9.15.y and < 9.12.y	
6.1	Tip of the tongue	7.1.x		
6.2	Center of the tongue body	7.1.x		
6.3	Left border of the tongue			6.2.z
6.4	Right border of the tongue			6.2.z
7.1	top of spine (center of head rotation)			
8.1	Middle point of outer upper lip contour	7.1.x		
8.2	Middle point of outer lower lip contour	7.1.x		
8.3	Left corner of outer lip contour			
8.4	Right corner of outer lip contour			
8.5	Midpoint between f.p. 8.3 and 8.1 in outer upper lip contour	(8.3.x+8.1.x)/2		
8.6	Midpoint between f.p. 8.4 and 8.1 in outer upper lip contour	(8.4.x+8.1.x)/2		
8.7	Midpoint between f.p. 8.3 and 8.2 in outer lower lip contour	(8.3.x+8.2.x)/2		
8.8	Midpoint between f.p. 8.4 and 8.2 in outer lower lip contour	(8.4.x+8.2.x)/2		
8.9	Right hiph point of Cupid’s bow			
8.10	Left hiph point of Cupid’s bow			
9.1	Left nostril border			
9.2	Right nostril border			
9.3	Nose tip	7.1.x		
9.4	Bottom right edge of nose			
9.5	Bottom left edge of nose			
9.6	Right upper edge of nose bone			
9.7	Left upper edge of nose bone			
9.8	Top of the upper teeth	7.1.x		
9.9	Bottom of the lower teeth	7.1.x		
9.10	Bottom of the upper teeth	7.1.x		
9.11	Top of the lower teeth	7.1.x		
9.12	Middle lower edge of nose bone (or nose bump)	7.1.x	(9.6.y + 9.3.y)/2 or nose bump	
9.13	Left lower edge of nose bone		(9.6.y +9.3.y)/2	
9.14	Right lower edge of nose bone		(9.6.y +9.3.y)/2	
9.15	Bottom middle edge of nose	7.1.x		
10.1	Top of left ear			
10.2	Top of right ear			
10.3	Back of left ear		(10.1.y+10.5.y)/2	
10.4	Back of right ear		(10.2.y+10.6.y)/2	
10.5	Bottom of left ear lobe			
10.6	Bottom of right ear lobe			

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.7	Lower contact point between left lobe and face			
10.8	Lower contact point between right lobe and face			
10.9	Upper contact point between left ear and face			
10.10	Upper contact point between right ear and face			
11.1	Middle border between hair and forehead	7.1.x		
11.2	Right border between hair and forehead	< 4.4.x		
11.3	Left border between hair and forehead	> 4.3.x		
11.4	Top of skull	7.1.x		> 10.4.z and < 10.2.z
11.5	Hair thickness over f.p. 11.4	11.4.x		11.4.z
11.6	Back of skull	7.1.x	3.5.y	

Table C.4 — FDP fields

FDP field	Description
featurePointsCoord	contains a **Coordinate** node. Specifies feature points for the calibration of the proprietary face. The coordinates are listed in the 'point' field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label (e.g. feature point 3.14 before feature point 4.1).
textureCoords	contains a **Coordinate** node. Specifies texture coordinates for the feature points. The coordinates are listed in the **point** field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label (e.g. feature point 3.14 before feature point 4.1).
textureType	contains a hint to the decoder on the type of texture image, in order to allow better interpolation of texture coordinates for the vertices that are not feature points. If **textureType** is 0, the decoder should assume that the texture image is obtained by cylindrical projection of the face. If **textureType** is 1, the decoder should assume that the texture image is obtained by orthographic projection of the face.
faceDefTables	contains **faceDefTables** nodes. The behavior of FAPs is defined in this field for the face in **faceSceneGraph**.
faceSceneGraph	contains a **Group** node. In case of option 1, this can be used to contain a texture image as explained above. In case of option 2, this is the grouping node for face model rendered in the compositor and has to contain the face model. In this case, the effect of Facial Animation Parameters is defined in the **faceDefTablesfield**.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Table C.5 — Values for viseme_select

viseme_select	phonemes	example
0	none	na
1	p, b, m	put, bed, mill
2	f, v	far, voice
3	T,D	think, that
4	t, d	tip, doll
5	k, g	call, gas
6	tS, dZ, S	chair, join, she
7	s, z	sir, zeal
8	n, l	lot, not
9	r	red
10	A:	car
11	e	bed
12	I	tip
13	Q	top
14	U	book

The symbolic constants and variables used in the syntax diagrams are listed in the table below. Every group and BAP is assumed to have a unique unsigned integer value assignment.

TABLE:

Name	Definition	Description
NBAP_GROUP	Array[1..BAP_NUM_GROUPS]	Number of baps in each group.
BAPS_IN_GROUP	Array[1..BAP_NUM_GROUPS] [1..MAX_BAPS]	List of BAPs belonging to each group

Name	Value	Definition
MAX_BAPS	22	Maximum number of BAPs that belong to any group
BAP_NUM_GROUPS	24	Total number of BAP groups
NUM_BAPS	296	Total number of BAPs

The detailed BAPs and the BAP groups are given below:

BAP ID	BAP NAME	DESCRIPTION	Quanti-sation step size	Min/Max I-Frame quantised values	Min/Max P-Frame quantised values
1	sacroiliac_tilt	Forward-backward motion of the pelvis in the sagittal plane	64	-960/ +960	-600/ +600
2	sacroiliac_torsion	Rotation of the pelvis along the body vertical axis (defined by skeleton root)	64	-960/ +960	-600/ +600

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3	sacroiliac_roll	Side to side swinging of the pelvis in the coronal plane	64	-960/ +960	-600/ +600
4	l_hip_flexion	Forward-backward rotation in the sagittal plane	128	-1260/ +1260	-600/ +600
5	r_hip_flexion	Forward-backward rotation in the sagittal plane	128	-1260/ +1260	-600/ +600
6	l_hip_abduct	Sideward opening in the coronal plane	128	-960/ +480	-600/ +600
7	r_hip_abduct	Sideward opening in the coronal plane	128	-960/ +480	-600/ +600
8	l_hip_twisting	Rotation along the thigh axis	256	-960/ +960	-360/ +360
9	r_hip_twisting	Rotation along the thigh axis	256	-960/ +960	-360/ +360
10	l_knee_flexion	Flexion-extension of the leg in the sagittal plane	128	-1500/ +180	-600/ +600
11	r_knee_flexion	Flexion-extension of the leg in the sagittal plane	128	-1500/ +180	-600/ +600
12	l_knee_twisting	Rotation along the shank axis.	256	-960/ +960	-360/ +360
13	r_knee_twisting	Rotation along the shank axis.	256	-960/ +960	-360/ +360
14	l_ankle_flexion	Flexion-extension of the foot in the sagittal plane	128	-780/ +780	-600/ +600
15	r_ankle_flexion	Flexion-extension of the foot in the sagittal plane	128	-780/ +780	-600/ +600
16	l_ankle_twisting	Rotation along the knee axis	256	-960/ +960	-360/ +360
17	r_ankle_twisting	Rotation along the knee axis	256	-960/ +960	-360/ +360
18	l_subtalar_flexion	Sideward orientation of the foot	256	-780/ +780	-600/ +600
19	r_subtalar_flexion	Sideward orientation of the foot	256	-780/ +780	-600/ +600
20	l_midtarsal_twisting	Internal twisting of the foot (also called navicular joint in anatomy)	256	-180/ +180	-120/ +120
21	r_midtarsal_ twisting	Internal twisting of the foot (also called navicular joint in anatomy)	256	-180/ +180	-120/ +120
22	l_metatarsal_flexion	Up and down rotation of the toe in the sagittal plane	256	-780/ +780	-600/ +600
23	r_metatarsal_flexion	Up and down rotation of the toe in the sagittal plane	256	-780/ +780	-600/ +600
24	l_sternoclavicular_abduct	Up and down motion in the coronal plane	128	-60/ +240	-120/ +120
25	r_sternoclavicular_abduct	Up and down motion in the coronal plane	128	-240/ +60	-120/ +120
26	l_sternoclavicular_rotate	Rotation in the transverse plane	128	-120/ +120	-60/ +60

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

27	r_sternoclavicular_rotate	Rotation in the transverse plane	128	-120/ +120	-60/ +60
28	l_acromioclavicular_abduct	Up and down motion in the coronal plane	128	-60/ +360	-120/ +120
29	r_acromioclavicular_abduct	Up and down motion in the coronal plane	128	-360/ +60	-120/ +120
30	l_acromioclavicular_rotate	Rotation in the transverse plane	128	-360/ +360	-120/ +120
31	r_acromioclavicular_rotate	Rotation in the transverse plane	128	-360/ +360	-120/ +120
32	l_shoulder_flexion	Forward-backward motion in the sagittal plane	64	-1080/ +1860	-600/ +600
33	r_shoulder_flexion	Forward-backward motion in the sagittal plane	64	-1080/ +1860	-600/ +600
34	l_shoulder_abduct	Sideward motion in the coronal plane	64	-240/ +1860	-600/ +600
35	r_shoulder_abduct	Sideward motion in the coronal plane	64	-1860/ +240	-600/ +600
36	l_shoulder_twisting	Rotation along the scapular axis	256	-960/ +960	-360/ +360
37	r_shoulder_twisting	Rotation along the scapular axis	256	-960/ +960	-360/ +360
38	l_elbow_flexion	Flexion-extension of the arm in the sagittal plane	64	-60/ +1560	-600/ +600
39	r_elbow_flexion	Flexion-extension of the arm in the sagittal plane	64	-60/ +1560	-600/ +600
40	l_elbow_twisting	Rotation of the forearm along the upper arm axis.	256	-960/ +960	-360/ +360
41	r_elbow_twisting	Rotation of the forearm along the upper arm axis.	256	-960/ +960	-360/ +360
42	l_wrist_flexion	Rotation of the hand in the coronal plane	128	-960/ +960	-600/ +600
43	r_wrist_flexion	Rotation of the hand in the coronal plane	128	-960/ +960	-600/ +600
44	l_wrist_pivot	Rotation of the hand in the sagittal planes	128	-660/ +660	-360/ +360
45	r_wrist_pivot	Rotation of the hand in the sagittal planes	128	-660/ +660	-360/ +360
46	l_wrist_twisting	Rotation of the hand along the forearm axis	256	-660/ +660	-360/ +360
47	r_wrist_twisting	Rotation of the hand along the forearm axis	256	-660/ +660	-360/ +360
48	skullbase_roll	Sideward motion of the skull along the frontal axis	128	-1860/ +1860	-600/ +600
49	skullbase_torsion	Twisting of the skull along the vertical axis	128	-1860/ +1860	-600/ +600
50	skullbase_tilt	Forward-backward motion in the sagittal plane along a lateral axis	128	-1860/ +1860	-600/ +600
51	vc1_roll	Sideward motion of vertebra C1	256	-240/ +240	-120/ +120

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

52	vc1_torsion	Twisting of vertebra C1	256	-240/ +240	-120/ +120
53	vc1_tilt	Forward-backward motion of vertebra C1 in the sagittal plane	256	-240/ +240	-120/ +120
54	vc2_roll	Sideward motion of vertebra C2	128	-660/ +660	-360/ +360
55	vc2_torsion	Twisting of vertebra C2	128	-660/ +660	-360/ +360
56	vc2_tilt	Forward-backward motion of vertebra C2 in the sagittal plane	128	-660/ +660	-360/ +360
57	vc3_roll	Sideward motion of vertebra C3	256	-240/ +240	-120/ +120
58	vc3_torsion	Twisting of vertebra C3	256	-240/ +240	-120/ +120
59	vc3_tilt	Forward-backward motion of vertebra C3 in the sagittal plane	256	-240/ +240	-120/ +120
60	vc4_roll	Sideward motion of vertebra C4	128	-660/ +660	-360/ +360
61	vc4_torsion	Twisting of vertebra C4	128	-660/ +660	-360/ +360
62	vc4_tilt	Forward-backward motion of vertebra C4 in the sagittal plane	128	-660/ +660	-360/ +360
63	vc5_roll	Sideward motion of vertebra C5	256	-240/ +240	-120/ +120
64	vc5_torsion	Twisting of vertebra C5	256	-240/ +240	-120/ +120
65	vc5_tilt	Forward-backward motion of vertebra C5 in the sagittal plane	256	-240/ +240	-120/ +120
66	vc6_roll	Sideward motion of vertebra C6	256	-240/ +240	-120/ +120
67	vc6_torsion	Twisting of vertebra C6	256	-240/ +240	-120/ +120
68	vc6_tilt	Forward-backward motion of vertebra C6 in the sagittal plane	256	-240/ +240	-120/ +120
69	vc7_roll	Sideward motion of vertebra C7	256	-240/ +240	-120/ +120
70	vc7_torsion	Twisting of vertebra C7	256	-240/ +240	-120/ +120
71	vc7_tilt	Forward-backward motion of vertebra C7 in the sagittal plane	256	-240/ +240	-120/ +120
72	vt1_roll	Sideward motion of vertebra T1	128	-660/ +660	-360/ +360
73	vt1_torsion	Twisting of vertebra T1	128	-660/ +660	-360/ +360
74	vt1_tilt	Forward-backward motion of vertebra T1 in the sagittal plane	128	-660/ +660	-360/ +360
75	vt2_roll	Sideward motion of vertebra T2	256	-240/ +240	-120/ +120
76	vt2_torsion	Twisting of vertebra T2	256	-240/ +240	-120/ +120

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

77	vt2_tilt	Forward-backward motion of vertebra T2 in the sagittal plane	256	-240/ +240	-120/ +120
78	vt3_roll	Sideward motion of vertebra T3	256	-240/ +240	-120/ +120
79	vt3_torsion	Twisting of vertebra T3	256	-240/ +240	-120/ +120
80	vt3_tilt	Forward-backward motion of vertebra T3 in the sagittal plane	256	-240/ +240	-120/ +120
81	vt4_roll	Sideward motion of vertebra T4	256	-240/ +240	-120/ +120
82	vt4_torsion	Twisting of vertebra T4	256	-240/ +240	-120/ +120
83	vt4_tilt	Forward-backward motion of vertebra T4 in the sagittal plane	256	-240/ +240	-120/ +120
84	vt5_roll	Sideward motion of vertebra T5	256	-240/ +240	-120/ +120
85	vt5_torsion	Twisting of vertebra T5	256	-240/ +240	-120/ +120
86	vt5_tilt	Forward-backward motion of vertebra T5 in the sagittal plane	256	-240/ +240	-120/ +120
87	vt6_roll	Sideward motion of vertebra T6	128	-660/ +660	-360/ +360
88	vt6_torsion	Twisting of vertebra T6	128	-660/ +660	-360/ +360
89	vt6_tilt	Forward-backward motion of vertebra T6 in the sagittal plane	128	-660/ +660	-360/ +360
90	vt7_roll	Sideward motion of vertebra T7	256	-240/ +240	-120/ +120
91	vt7_torsion	Twisting of vertebra T7	256	-240/ +240	-120/ +120
92	vt7_tilt	Forward-backward motion of vertebra T7 in the sagittal plane	256	-240/ +240	-120/ +120
93	vt8_roll	Sideward motion of vertebra T8	256	-240/ +240	-120/ +120
94	vt8_torsion	Twisting of vertebra T8	256	-240/ +240	-120/ +120
95	vt8_tilt	Forward-backward motion of vertebra T8 in the sagittal plane	256	-240/ +240	-120/ +120
96	vt9_roll	Sideward motion of vertebra T9	256	-240/ +240	-120/ +120
97	vt9_torsion	Twisting of vertebra T9	256	-240/ +240	-120/ +120
98	vt9_tilt	Forward-backward motion of vertebra T9 in the sagittal plane	256	-240/ +240	-120/ +120
99	vt_10roll	Sideward motion of vertebra T10	128	-660/ +660	-360/ +360
100	vt10_torsion	Twisting of vertebra T10	128	-660/ +660	-360/ +360
101	vt10_tilt	Forward-backward motion of vertebra T10 in sagittal plane	128	-660/ +660	-360/ +360

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

102	vt11_roll	Sideward motion of vertebra T11	256	-240/ +240	-120/ +120
103	vt11_torsion	Twisting of vertebra T11	256	-240/ +240	-120/ +120
104	vt11_tilt	Forward-backward motion of vertebra T11 in sagittal plane	256	-240/ +240	-120/ +120
105	vt12_roll	Sideward motion of vertebra T12	256	-240/ +240	-120/ +120
106	vt12_torsion	Twisting of vertebra T12	256	-240/ +240	-120/ +120
107	vt12_tilt	Forward-backward motion of vertebra T12 in sagittal plane	256	-240/ +240	-120/ +120
108	vl1_roll	Sideward motion of vertebra L1	128	-660/ +660	-360/ +360
109	vl1_torsion	Twisting of vertebra L1	128	-660/ +660	-360/ +360
110	vl1_tilt	Forward-backward motion of vertebra L1 in sagittal plane	128	-660/ +660	-360/ +360
111	vl2_roll	Sideward motion of vertebra L2	256	-240/ +240	-120/ +120
112	vl2_torsion	Twisting of vertebra L2	256	-240/ +240	-120/ +120
113	vl2_tilt	Forward-backward motion of vertebra L2 in sagittal plane	256	-240/ +240	-120/ +120
114	vl3_roll	Sideward motion of vertebra L3	128	-660/ +660	-360/ +360
115	vl3_torsion	Twisting of vertebra L3	128	-660/ +660	-360/ +360
116	vl3_tilt	Forward-backward motion of vertebra L3 in sagittal plane	128	-660/ +660	-360/ +360
117	vl4_roll	Sideward motion of vertebra L4	256	-240/ +240	-120/ +120
118	vl4_torsion	Twisting of vertebra L4	256	-240/ +240	-120/ +120
119	vl4_tilt	Forward-backward motion of vertebra L4 in sagittal plane	256	-240/ +240	-120/ +120
120	vl5_roll	Sideward motion of vertebra L5	128	-660/ +660	-360/ +360
121	vl5_torsion	Twisting of vertebra L5	128	-660/ +660	-360/ +360
122	vl5_tilt	Forward-backward motion of vertebra L5 in sagittal plane	128	-660/ +660	-360/ +360
123	l_pinky0_flexion	Metacarpal flexing mobility of the pinky finger	512	-240/ +240	-120/ +120
124	r_pinky0_flexion	Metacarpal flexing mobility of the pinky finger	512	-240/ +240	-120/ +120
125	l_pinky1_flexion	First knukle of the pinky finger	128	-1140/ +240	-240/ +240
126	r_pinky1_flexion	First knukle of the pinky finger	128	-1140/ +240	-240/ +240

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

127	l_pinky1_pivot	Lateral mobility of the pinky finger	128	-420/ +120	-120/ +120
128	r_pinky1_pivot	Lateral mobility of the pinky finger	128	-120/ +420	-120/ +120
129	l_pinky1_twisting	Along the pinky finger axis	256	-180/ +360	-120/ +120
130	r_pinky1_twisting	Along the pinky finger axis	256	-360/ +180	-120/ +120
131	l_pinky2_flexion	Second knuckle of the pinky number	128	-1380/ +60	-240/ +240
132	r_pinky2_flexion	Second knuckle of the pinky number	128	-1380/ +60	-240/ +240
133	l_pinky3_flexion	Third knuckle of the pinky finger	128	-960/ +60	-240/ +240
134	r_pinky3_flexion	Third knuckle of the pinky finger	128	-960/ +60	-240/ +240
135	l_ring0_flexion	Metacarpal flexing mobility of the ring finger	256	-180/ +180	-120/ +120
136	r_ring0_flexion	Metacarpal flexing mobility of the ring finger	256	-180/ +180	-120/ +120
137	l_ring1_flexion	First knukle of the ring finger	128	-1140/ +360	-240/ +240
138	r_ring1_flexion	First knukle of the ring finger	128	-1140/ +360	-240/ +240
139	l_ring1_pivot	Lateral mobility of the ring finger	128	-240/ +120	-120/ +120
140	r_ring1_pivot	Lateral mobility of the ring finger	128	-120/ +240	-120/ +120
141	l_ring1_twisting	Along the ring finger axis	256	-240/ +240	-120/ +120
142	r_ring1_twisting	Along the ring finger axis	256	-240/ +240	-120/ +120
143	l_ring2_flexion	Second knuckle of the ring number	128	-1380/ +60	-240/ +240
144	r_ring2_flexion	Second knuckle of the ring number	128	-1380/ +60	-240/ +240
145	l_ring3_flexion	Third knuckle of the ring finger	128	-960/ +60	-240/ +240
146	r_ring3_flexion	Third knuckle of the ring finger	128	-960/ +60	-240/ +240
147	l_middle0_flexion	Metacarpal flexing mobility of the middle finger	256	-120/ +120	-120/ +120
148	r_middle0_flexion	Metacarpal flexing mobility of the middle finger	256	-120/ +120	-120/ +120
149	l_middle1_flexion	First knukle of the middle finger	128	-1140/ +360	-240/ +240
150	r_middle1_flexion	First knukle of the middle finger	128	-1140/ +360	-240/ +240
151	l_middle1_pivot	Lateral mobility of the middle finger	128	-180/ +180	-120/ +120

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

152	r_middle1_pivot	Lateral mobility of the middle finger	128	-180/ +180	-120/ +120
153	l_middle1_twisting	Along the middle finger axis	256	-180/ +180	-120/ +120
154	r_middle1_twisting	Along the middle finger axis	256	-180/ +180	-120/ +120
155	l_middle2_flexion	Second knuckle of the middle number	128	-1380/ +60	-240/ +240
156	r_middle2_flexion	Second knuckle of the middle number	128	-1380/ +60	-240/ +240
157	l_middle3_flexion	Third knuckle of the middle finger	128	-960/ +60	-240/ +240
158	r_middle3_flexion	Third knuckle of the middle finger	128	-960/ +60	-240/ +240
159	l_index0_flexion	Metacarpal flexing mobility of the index finger	256	-60/ +60	-60/ +60
160	r_index0_flexion	Metacarpal flexing mobility of the index finger	256	-60/ +60	-60/ +60
161	l_index1_flexion	First knukle of the index finger	128	-1140/ +360	-240/ +240
162	r_index1_flexion	First knukle of the index finger	128	-1140/ +360	-240/ +240
163	l_index1_pivot	Lateral mobility of the index finger	128	-120/ +240	-60/ +60
164	r_index1_pivot	Lateral mobility of the index finger	128	-240/ +120	-60/ +60
165	l_index1_twisting	Along the index finger axis	256	-240/ +180	-120/ +120
166	r_index1_twisting	Along the index finger axis	256	-180/ +240	-120/ +120
167	l_index2_flexion	Second knuckle of the index number	128	-1380/ +60	-240/ +240
168	r_index2_flexion	Second knuckle of the index number	128	-1380/ +60	-240/ +240
169	l_index3_flexion	Third knuckle of the index finger	128	-960/ +60	-240/ +240
170	r_index3_flexion	Third knuckle of the index finger	128	-960/ +60	-240/ +240
171	l_thumb1_flexion	First knukle of the thumb finger	128	-480/ +960	-240/ +240
172	r_thumb1_flexion	First knukle of the thumb finger	128	-960/ +480	-240/ +240
173	l_thumb1_pivot	Lateral mobility of the thumb finger	128	-120/ +1080	-240/ +240
174	r_thumb1_pivot	Lateral mobility of the thumb finger	128	-1080/ +120	-240/ +240
175	l_thumb1_twisting	Along the thumb finger axis	256	-720/ +120	-240/ +240
176	r_thumb1_twisting	Along the thumb finger axis	256	-720/ +120	-240/ +240

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

177	l_thumb2_flexion	Second knuckle of the thumb number	128	-780/ +60	-240/ +240
178	r_thumb2_flexion	Second knuckle of the thumb number	128	-780/ +60	-240/ +240
179	l_thumb3_flexion	Third knuckle of the thumb finger	128	-960/ +60	-240/ +240
180	r_thumb3_flexion	Third knuckle of the thumb finger	128	-960/ +60	-240/ +240
181	HumanoidRoot_tr_vertical	Body origin translation in vertical direction	64	-1860/ +1860	-600/ +600
182	HumanoidRoot_tr_lateral	Body origin translation in lateral direction	64	-1860/ +1860	-600/ +600
183	HumanoidRoot_tr_frontal	Body origin translation in frontal direction	64	-1860/ +1860	-600/ +600
184	HumanoidRoot_rt_body_turn	Rotation of the skeleton root along the body coordinate system's vertical axis	64	-1860/ +1860	-600/ +600
185	HumanoidRoot_rt_body_roll	Rotation of the skeleton root along the body coordinate system's frontal axis	64	-1860/ +1860	-600/ +600
186	HumanoidRoot_rt_body_tilt	Rotation of the skeleton root along the body coordinate system's lateral axis	64	-1860/ +1860	-600/ +600

BAPs 187 through 296 are extension BAPs for use with the BodyDefTable nodes., see ISO/IEC 14496-1:2001 for specification of BodyDefTable nodes. These BAPs are user-defined. The step sizes of these BAPs are 1, and I-Frame and P-Frame default minmax values are not defined.

GROUP ID	GROUP NAME	BAPS
1	Pelvis	sacroiliac_tilt, sacroiliac_torsion, sacroiliac_roll (1,2,3)
2	Left leg1	l_hip_flexion, l_hip_abduct, l_knee_flexion, l_ankle_flexion (4,6,10,14)
3	Right leg1	r_hip_flexion, r_hip_abduct, r_knee_flexion, r_ankle_flexion (5,7,11,15)
4	Left leg2	l_hip_twisting, l_knee_twisting, l_ankle_twisting, l_subtalar_flexion, l_midtarsal_flexion, l_metatarsal_flexion (8,12,16,18,20,22)
5	Right leg2	r_hip_twisting, r_knee_twisting, r_ankle_twisting, r_subtalar_flexion, r_midtarsal_flexion, r_metatarsal_flexion (9,13,17,19,21,23)
6	Left arm1	l_shoulder_flexion, l_shoulder_abduct, l_shoulder_twisting, l_elbow_flexion, l_wrist_flexion (32,34,36,38,42)
7	Right arm1	r_shoulder_flexion, r_shoulder_abduct, r_shoulder_twisting, r_elbow_flexion, r_wrist_flexion (33,35,37,39,43)
8	Left arm2	l_sternoclavicular_abduct, l_sternoclavicular_rotate, l_acromioclavicular_abduct, l_acromioclavicular_rotate, l_elbow_twisting, l_wrist_pivot, l_wrist_twisting (24,26,28,30,40,44,46)

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9	Right arm2	r_sternoclavicular_abduct, r_sternoclavicular_rotate, r_acromioclavicular_abduct, r_acromioclavicular_rotate, r_elbow_twisting, r_wrist_pivot, r_wrist_twisting (25,27,29,31,41,45,47)
10	Spine1	skullbase_roll, skullbase_torsion, skullbase_tilt, vc4roll, vc4torsion, vc4tilt, vt6roll, vt6torsion, vt6tilt, vl3roll, vl3torsion, vl3tilt, (48,49,50,60,61,62,87,88,89,114,115,116)
11	Spine2	vc2roll, vc2torsion, vc2tilt, vt1roll, vt1torsion, vt1tilt, vt10roll, vt10torsion, vt10tilt, vl1roll, vl1torsion, vl1tilt, vl5roll, vl5torsion, vl5tilt (54,55,56,72,73,74,99,100,101,108, 109,110,120,121,122)
12	Spine3	vc3roll, vc3torsion, vc3tilt, vc6roll, vc6torsion, vc6tilt, vt4roll, vt4torsion, vt4tilt, vt8roll, vt8torsion, vt8tilt, vt12roll, vt12torsion, vt12tilt vl4roll, vl4torsion, vl4tilt, (57,58,59,66,67,68,81,82,83,93,94,95, 105,106,107,117,118,119)
13	Spine4	vc5roll, vc5torsion, vc5tilt, vc7roll, vc7torsion, vc7tilt vt2roll, vt2torsion, vt2tilt, vt7roll, vt7torsion, vt7tilt, vt11roll, vt11torsion, vt11tilt, vl2roll, vl2torsion, vl2tilt, (63,64,65,69,70,71,75,76,77,90,91, 92,102,103,104,111,112,113)
14	Spine5	vc1roll, vc1torsion, vc1tilt, vt3roll, vt3torsion, vt3tilt, vt5roll, vt5torsion, vt5tilt, vt9roll, vt9torsion, vt9tilt, (51,52,53,78,79,80,84,85,86,96,97,98)
15	Left hand1	l_pinky1_flexion, l_pinky2_flexion, l_pinky3_flexion, l_ring1_flexion, l_ring2_flexion, l_ring3_flexion, l_middle1_flexion, l_middle2_flexion,l_middle3_flexion, l_index1_flexion, l_index2_flexion, l_index3_flexion, l_thumb1_flexion, l_thumb1_pivot, l_thumb2_flexion, l_thumb3_flexion (125,131,133,137,143,145,149,155,157, 161,167,169,171,173,177,179)

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16	Right hand1	r_pinky1_flexion, r_pinky2_flexion, r_pinky3_flexion, r_ring1_flexion, r_ring2_flexion, r_ring3_flexion, r_middle1_flexion, r_middle2_flexion,r_middle3_flexion, r_index1_flexion, r_index2_flexion, r_index3_flexion, r_thumb1_flexion, r_thumb1_pivot, r_thumb2_flexion, r_thumb3_flexion (126,132,134,138,144,146,150,156,158, 162,168,170,172,174,178,180)
17	Left hand2	l_pinky0_flexion, l_pinky1_ pivot, l_pinky1_twisting, l_ring0_flexion, l_ring1_ pivot, l_ring1_twisting, l_middle0_flexion, l_middle1_ pivot, l_middle1_twisting, l_index0_flexion, l_index1_ pivot, l_index1_twisting, l_thumb1_twisting (123,127,129,135,139,141,147, 151,153,159,163,165,175)
18	Right hand2	r_pinky0_flexion, r_pinky1_ pivot, r_pinky1_twisting, r_ring0_flexion, r_ring1_ pivot, r_ring1_twisting, r_middle0_flexion, r_middle1_ pivot, r_middle1_twisting, r_index0_flexion, r_index1_ pivot, r_index1_twisting, r_thumb1_twisting (124,128,130,136,140,142,148, 152,154,160,164,166,176)
19	Global positioning	HumanoidRoot_tr_vertical, HumanoidRoot_tr_lateral, HumanoidRoot_tr_frontal, HumanoidRoot_rt_body_turn, HumanoidRoot_rt_body_roll, HumanoidRoot_rt_body_tilt (181,182,183,184,185,186)

Additionally, groups 20 to 24 contain extension BAPs. Extension BAPs are user-defined BAPs for use with BodyDefTables. See Systems CD for specification of BodyDefTable nodes. Groups 20 to 24 each contain 22 BAPs. Thus, the extension BAP groups are as follows:

GROUP ID	GROUP NAME	BAPs
20	Extension1	187…208
21	Extension2	209…230
22	Extension3	231…252
23	Extension4	253…274
24	Extension5	275…296

Detailed Degrees of Freedom

The ***degrees of freedom (DOF)*** sufficient to locate and animate a human body are decomposed into six DOF for the global location and 180 DOF for the internal mobilities. The topology of the suggested joints is given below. The hands are optional with an additional set of joints.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Suggested default joint center values for broadcast applications

HumanoidRoot	0.0000	0.9723	-0.0728
sacroiliac	0.0000	0.9710	-0.0728
l_hip	0.0956	0.9364	0.0000
l_knee	0.0956	0.5095	-0.0036
l_ankle	0.0946	0.0762	-0.0261
l_subtalar	0.0956	0.0398	0.0069
l_midtarsal	0.1079	0.0317	0.0670
l_metatarsal	0.0942	0.0092	0.1239
r_hip	-0.0956	0.9364	0.0000
r_knee	-0.0956	0.5095	-0.0036
r_ankle	-0.0946	0.0762	-0.0261
r_subtalar	-0.0956	0.0398	0.0069
r_midtarsal	-0.1079	0.0317	0.0670
r_metatarsal	-0.0942	0.0092	0.1239
vl5	0.0000	1.0817	-0.0728
vl4	0.0000	1.1174	-0.0727
vl3	0.0000	1.1525	-0.0727
vl2	0.0000	1.1795	-0.0727
vl1	0.0000	1.2161	-0.0727
vt12	0.0000	1.2527	-0.0727
vt11	0.0000	1.2918	-0.0727
vt10	0.0000	1.3098	-0.0737
vt9	0.0000	1.3375	-0.0752
vt8	0.0000	1.3631	-0.0758
vt7	0.0000	1.3875	-0.0745
vt6	0.0000	1.4116	-0.0712
vt5	0.0000	1.4351	-0.0657
vt4	0.0000	1.4569	-0.0587
vt3	0.0000	1.4832	-0.0482
vt2	0.0000	1.5011	-0.0397
vt1	0.0000	1.5201	-0.0300
vc7	0.0000	1.5382	-0.0213
vc6	0.0000	1.5607	-0.0073
vc5	0.0000	1.5770	-0.0012
vc4	0.0000	1.5912	-0.0014
vc3	0.0000	1.6050	-0.0033
vc2	0.0000	1.6178	-0.0033
vc1	0.0000	1.6394	0.0036
skullbase	0.0000	1.6440	0.0036
l_sternoclavicular	0.0757	1.4844	-0.0251
l_acromioclavicular	0.0899	1.4525	-0.0322
l_shoulder	0.1968	1.4642	-0.0265
l_elbow	0.1982	1.1622	-0.0557
l_wrist	0.1972	0.8929	-0.0690
l_thumb1	0.1912	0.8734	-0.0657
l_thumb2	0.1912	0.8156	-0.0079
l_thumb3	0.1912	0.8007	0.0070
l_index0	0.1912	0.8259	-0.0460

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

l_index1	0.1912	0.8050	-0.0460
l_index2	0.1912	0.7595	-0.0460
l_index3	0.1912	0.7370	-0.0460
l_middle0	0.1912	0.8282	-0.0710
l_middle1	0.1912	0.8071	-0.0710
l_middle2	0.1912	0.7525	-0.0710
l_middle3	0.1912	0.7263	-0.0710
l_ring0	0.1912	0.8302	-0.0972
l_ring1	0.1912	0.8098	-0.0972
l_ring2	0.1912	0.7570	-0.0972
l_ring3	0.1912	0.7328	-0.0972
l_pinky0	0.1912	0.8373	-0.1211
l_pinky1	0.1912	0.8173	-0.1211
l_pinky2	0.1912	0.7759	-0.1211
l_pinky3	0.1912	0.7584	-0.1211
r_sternoclavicular	-0.0757	1.4844	-0.0251
r_acromioclavicular	-0.0899	1.4525	-0.0322
r_shoulder	-0.1968	1.4642	-0.0265
r_elbow	-0.1982	1.1622	-0.0557
r_wrist	-0.1972	0.8929	-0.0690
r_thumb1	-0.1912	0.8734	-0.0657
r_thumb2	-0.1912	0.8156	-0.0079
r_thumb3	-0.1912	0.8007	0.0070
r_index0	-0.1912	0.8259	-0.0460
r_index1	-0.1912	0.8050	-0.0460
r_index2	-0.1912	0.7595	-0.0460
r_index3	-0.1912	0.7370	-0.0460
r_middle0	-0.1912	0.8282	-0.0710
r_middle1	-0.1912	0.8071	-0.0710
r_middle2	-0.1912	0.7525	-0.0710
r_middle3	-0.1912	0.7263	-0.0710
r_ring0	-0.1912	0.8302	-0.0972
r_ring1	-0.1912	0.8098	-0.0972
r_ring2	-0.1912	0.7570	-0.0972
r_ring3	-0.1912	0.7328	-0.0972
r_pinky0	-0.1912	0.8373	-0.1211
r_pinky1	-0.1912	0.8173	-0.1211
r_pinky2	-0.1912	0.7759	-0.1211
r_pinky3	-0.1912	0.7584	-0.1211

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

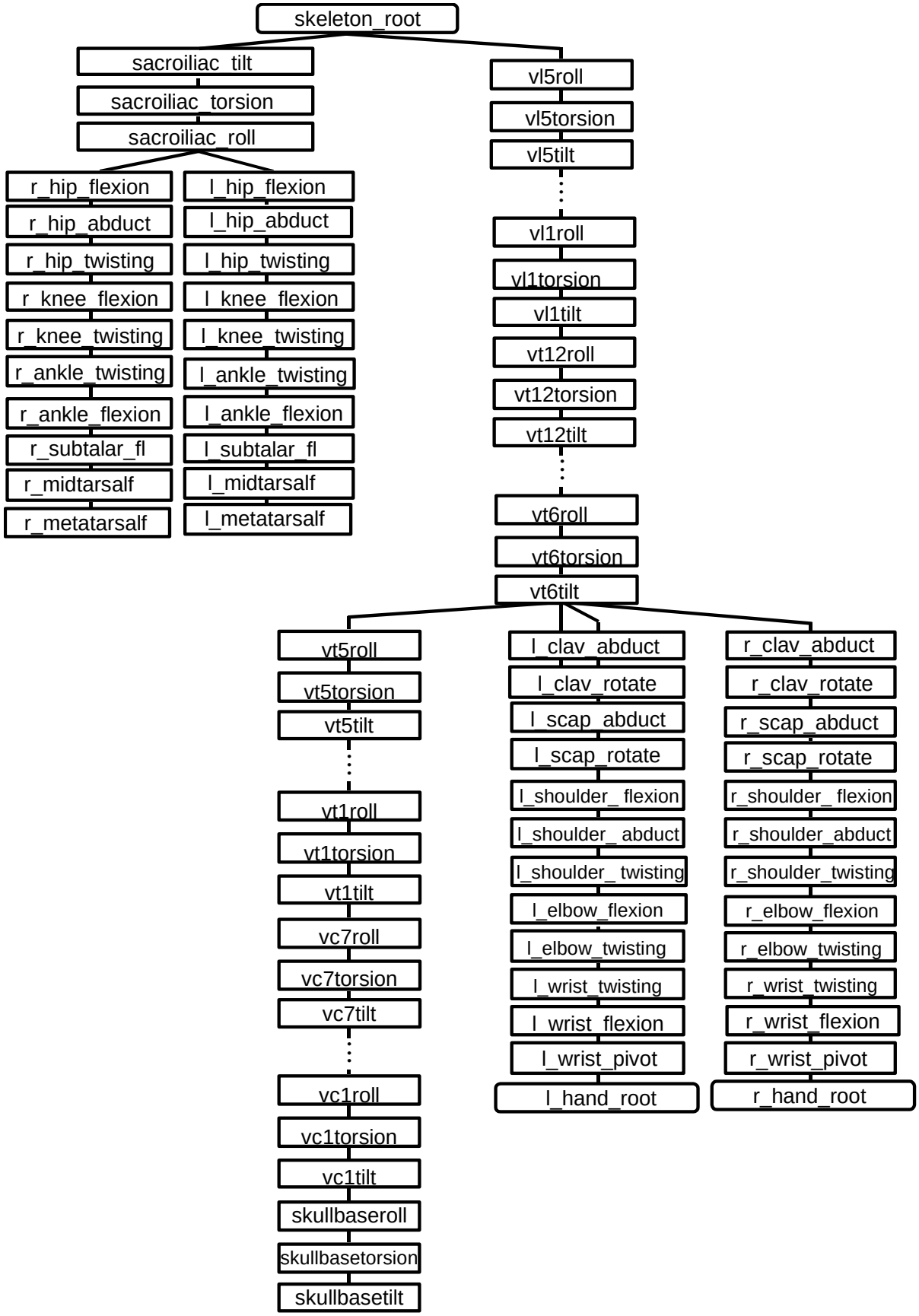


Figure C.2 — Body topology

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Lower Body

From the skeleton_root node, three DOF allow flexible pelvic orientation followed by nine DOF per leg, from the hip joint to the toe joint.

First, the pelvic mobilities are very important to convey a gender personification to the motion. The naming convention of these mobilities is also used for the spine DOF. The degrees of freedom are:

sacroiliac_tilt : forward-backward motion in the sagital plane

sacroiliac_torsion : rotation along the body vertical axis (defined by skeleton root)

sacroiliac_roll : side to side swinging in the coronal plane

The leg mobilities follow in this orde :

At the hip :

hip_flexion : forward-backward rotation in the sagital plane

hip_abduct : sideward opening in the coronal plane

hip_twisting : rotation along the thigh axis.

At the knee:

knee_flexion : flexion-extension of the leg in the sagital plane

knee_twisting : rotation along the shank axis.

At the ankle:

ankle_twisting : rotation along the shank axis. This joint is redundant with the knee_twisting except that only the foot rotate and not the shank segment.

ankle_flexion : flexion-extension of the foot in the sagital plane

At the foot complex:

The foot complex region is completely described with three degrees of freedom with independent position and orientation : the subtalar joint, the *mid_foot* joint, between the subtalar joint and the toe joint to capture complex internal relative movement of the foot bones (also called the navicular joint in the literature).

subtalar_flexion : sideward orientation of the foot

midtarsal_flexion : internal twisting of the foot (also called navicular joint in anatomy)

metatorsal_flexion : up and down rotation of the toe in the sagital plane

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

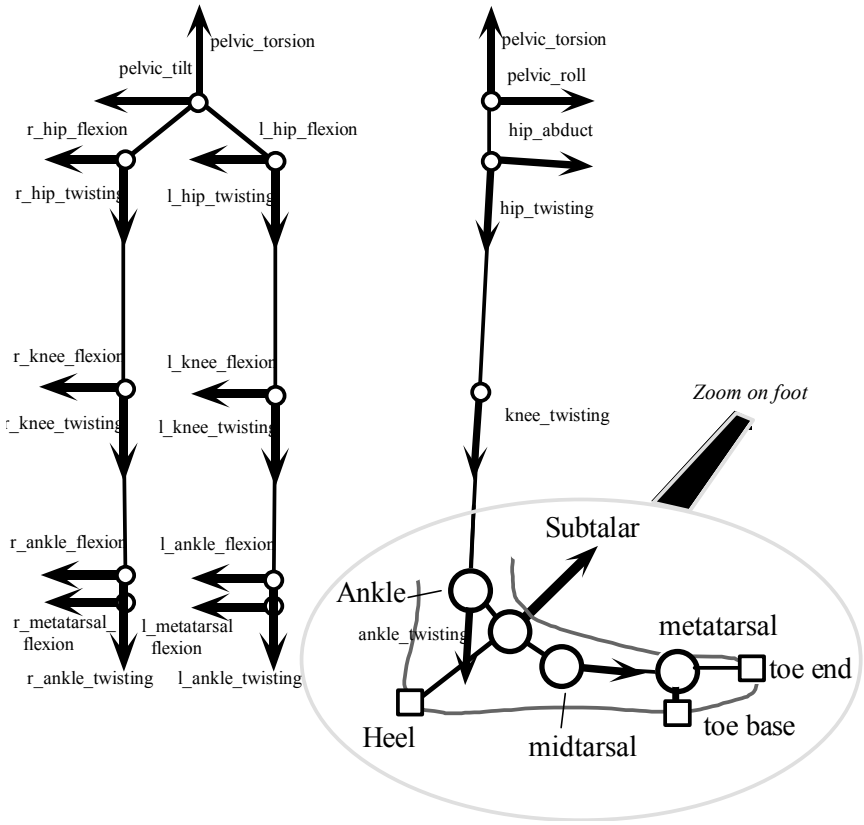


Figure C.3 — Front and side views of the mobilities of the leg

Upper Body

The suggested upper body degrees of freedom and their corresponding axes of rotation are shown in the following diagrams.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

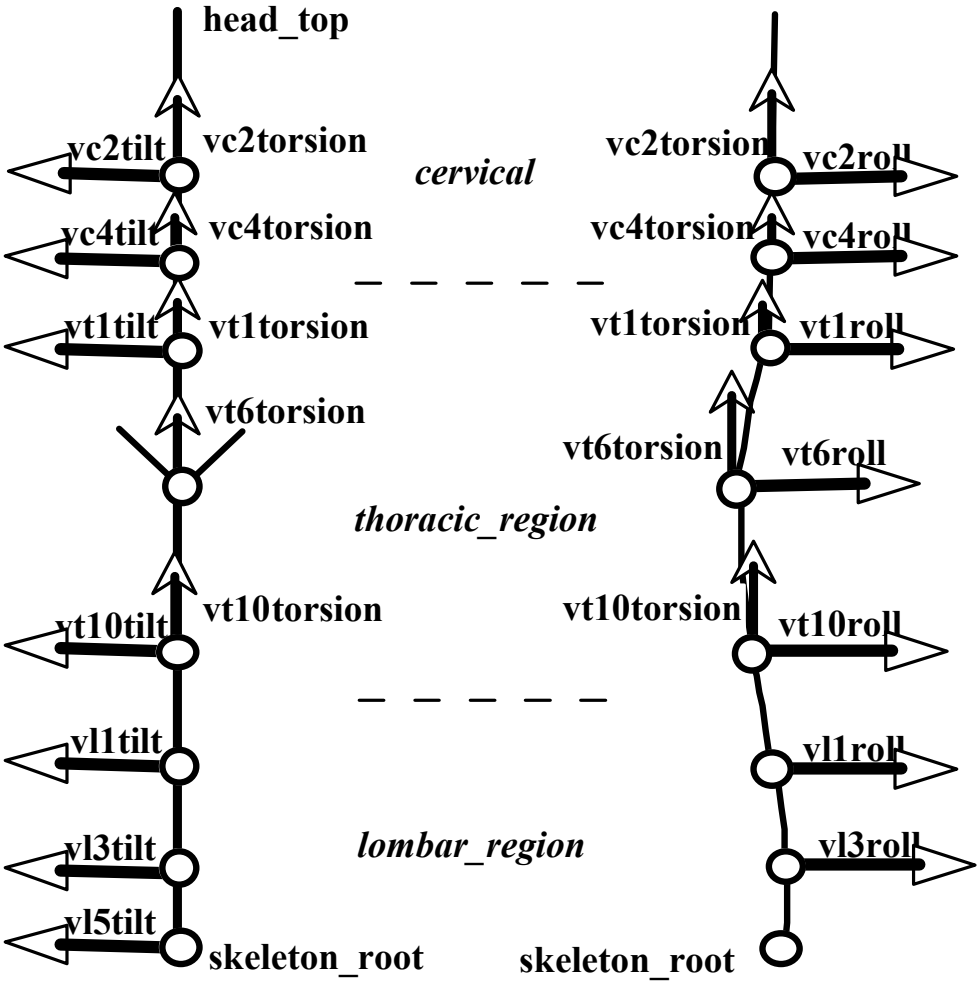


Figure C.4 — Front and side views of the mobilities of the simple spine

At the spine:

The vertebrae are dispatched into 5 *lumbar*, 12 *thoracic* and *cervical* groups.A total of 72 DOFs are defined for the spine, therefore complex applications with whole spine mobilities can be developed. However, typically, the application will use simpler spines that balance the computational speed, with the realism of animation. Therefore, 5 groups of spine are defined, from the simplest to most complex. It is suggested that the spine groups (Spine1, Spine2) are used for simple spine.

At the arms:

The arm structure is attached to the spine. The mobilities are similar to the ones defined for the leg when the arm is twisted such that the hand palm is facing backward. The name of the DOF respects this structure similarity.

Strictly speaking, the two clavicula DOF and the two scapula DOF are not part of the arm structure, they only initiate its articulated chain. The scapula joints improve the mechanical model of the shoulder complex. It should be noted that such a chain representation is only a step toward a mechanically correct representation of this region, mainly because the shoulder complex is in fact a closed loop mechanism. The scapula holds the same mobility as the clavicula but very close to the shoulder joint.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

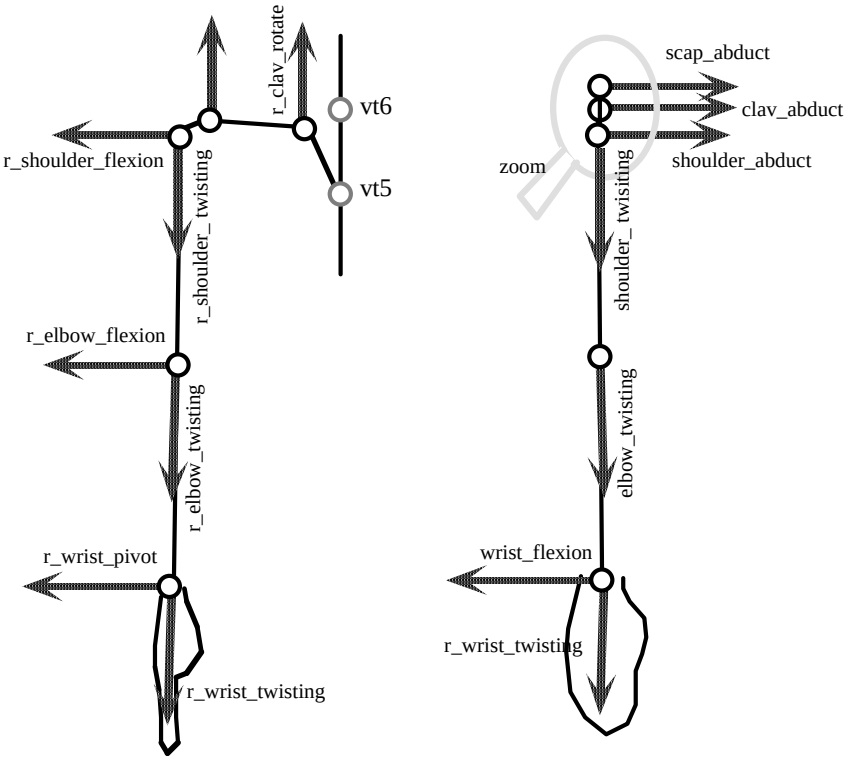


Figure C.5 — Front and side views of the mobilities of the arm in rest position

At the clavicula:

sternoclavicular_abduct : up and down motion in the coronal plane

sternoclavicular_rotate : rotation in the transverse plane

At the scapula:

acromioclavicular_abduct : up and down motion in the coronal plane

acromioclavicular_rotate : rotation in the transverse plane

At the shoulder:

shoulder_flexion : forward-backward motion in the sagital plane

shoulder_abduct : sideward motion in the coronal plane

shoulder_twisting : along the forearm axis

At the elbow:

elbow_flexion : flexion-extension of the arm in the sagital plane

elbow_twisting : along the arm axis.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

At the wrist:

wrist_twisting : along the arm axis. This DOF is redundant with the elbow twisting except that only the hand rotate and not the forearm

wrist_flexion : rotation of the hand in the coronal plane

wrist_pivot : rotation of the hand in the sagital planes

Head rotations

Note that there are three BAPs defined for head rotation: skullbase_roll, skullbase_torsion, skullbase_tilt. There are also 3 FAPs for head rotation. If both FAPs and BAPs are used for head rotation, then these FAPs shall denote the head rotation with respect to the skullbase coordinate system.

Hands

The hand mobilities have a standard structure for the five fingers. This structure is organized as:

a **flexing** DOF for closing the first knuckle

a **pivoting** rotation for the lateral mobility of the finger

a **twisting** rotation for small adjustments of the finger grasping orientation

two other **flexing** DOF for closing the second and third knuckles

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

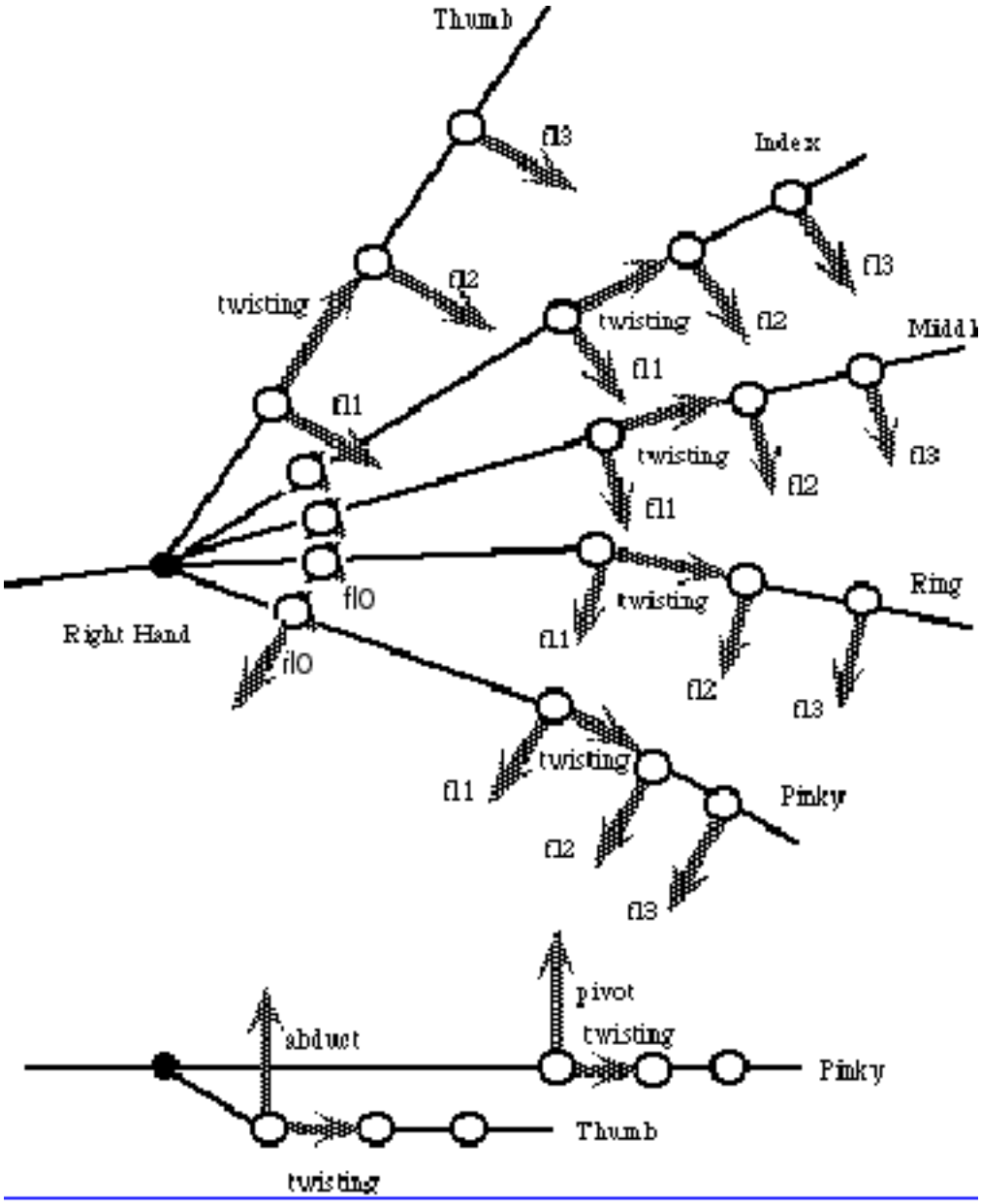


Figure C.6 — Mobilities of the right hand

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

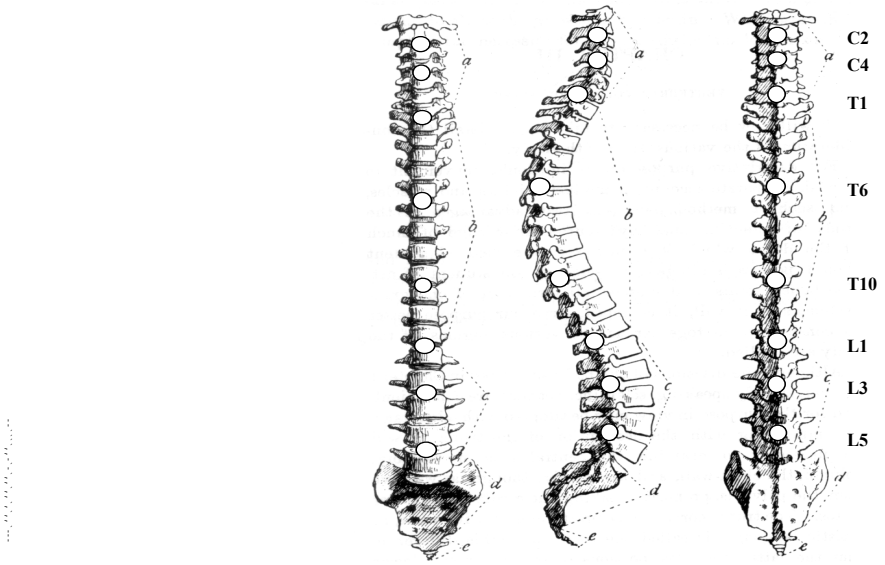


Figure C.7 — Example spine mobilities

Integration of Facial Animation with TTS

The following figure shows the complete block diagram describing the integration of a proprietary TTS Synthesizer into an ISO/IEC 14496 face animation system. The FAP bookmarks defined by the user in the input text of the TTS Stream are identified by the speech synthesizer and transmitted in ASCII format to the Phoneme to FAP converter using the TtsFAPInterface as defined in ISO/IEC 14496-3:2001.

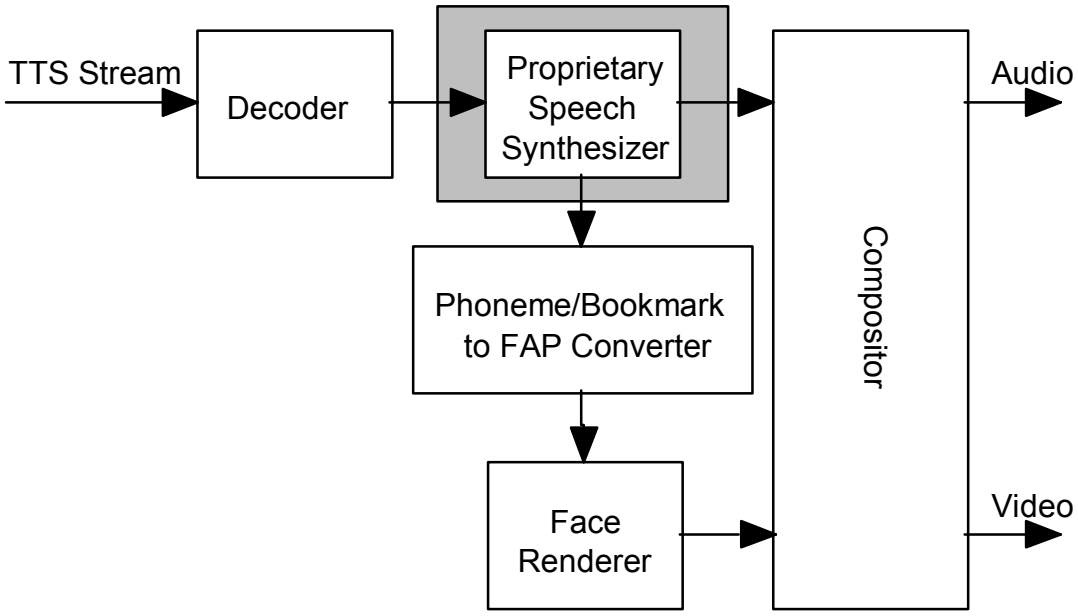


Figure C.8 — Blockdiagram showing the integration of a proprietary TTS into an ISO/IEC 14496 face animation system with Phoneme/Bookmark to FAP Converter.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The syntax of the bookmark sequences used to convey commands to the TTS system is the following:

<FAP *n* FAPfields *T C* >
n: FAP number defined according to annex C,Table C.1 with 2<=n<=68.
FAPfields := expression_select1* expression_intensity1 expression_select2* expression_intensity2, in case n == 2
FAPfields := *a***, in case 2 < n < =68
T: transition time defined in ms
C: time curve for computation of *a* during transition time*T*

where ":=" indicates a production rule

* defined according to Table C.3 (expression select)

** defined in units according to Table C.1

Phoneme/Bookmark to FAP Converter

In addition to its function to convert phonemes into visemes, the Phoneme/Bookmark to FAP Converter (Fig. 1) is responsible for translating the FAP bookmarks defined by the user and made available by the bookmark field of the TtsFAPInterface as defined in ISO/IEC 14496-1:2001 into a sequence of FAPs that can be interpreted by the Face Renderer, which then has to use them. An interpolation function is used for computing the FAP amplitudes. The converter merges the FAPs it derives from phonemes and bookmarks into one set of FAP parameters for each time instant using the following steps:

1. Set all low-level FAPs to interpolate.
2. Set specified FAPs to values computed using the interpolation function and phonemes.
3. init_face is set according to the following suggestion: In case of FAP 1 viseme_select1/2 != 0 and FAP 2 expression_select1/2 !=0, init_face=1; In case of FAP 2 expression_select1/2 !=0 and using FAP 3-68 for visemes, init_face=0.

Facial expressions and visemes are combined such that mouth closures are achieved for the relevant visemes while maintaining the overall facial expression. In case that the bookmarks contain visemes, they are ignored.

Transitions between facial expressions are achieved by computing the interpolation function for expression_intensity1 and expression_intensity2. Smooth transitions have to be achieved if expression_select1 and expression_select2 are the same for consecutive bookmarks.

Face Renderer:

In the case that the face model is also animated from an FAP parameter stream, the face renderer has to animate the face using the input from the TtsFAPInterface which is processed by the Phoneme/Bookmark to FAP converter and the input from the FAP decoder. In case that the renderer receives values for the same FAP from both sources, the values derived from the FAP stream take precedence.

Interpolation Functions

For simplicity, the following description on how to compute the amplitude of an FAP is explained using the amplitude *a*. The same method is applied to expression_intensity1 and expression_intensity2.

The FAP amplitude *a* defines the amplitude to be applied at the end of the transition time *T*. The amplitude a_s of the FAP at the beginning of the transition depends on previous bookmarks and can be equal to:

— 0 if the FAP bookmark is the first one with this FAP *n* made available through the TtsFAPInterface.

— *a* of the previous FAP bookmark with the same FAP *n* if a time longer than the previous transition time *T* has elapsed between these two FAP bookmarks.

— The actual reached amplitude due to the previous FAP definition if a time shorter than the previous transition time *T* has elapsed between the two FAP bookmarks.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

At the end of the transition time *T*, *a* is maintained until another FAP bookmark gives a new value to reach. To reset an FAP, a bookmark for FAP *n* with *a*=0 is transmitted in the text.

To avoid too many parameters for defining the evolution of the amplitude during the transition time, the functions that compute for each frame the amplitude of the FAP to be sent to the face renderer are predefined. Assuming that the transition time *T* is always 1, the following 3 functions *f(t)* are selected according the value of *C*:

$$f(t) = a_s + (a - a_s)t \quad \text{(linear) (1)}$$

$$f(t) = a_s + \begin{cases} (a - a_s)2t & for\ t \le 0.5 \\ (a - a_s)2(1-t) & for\ 0.5 < t \le 1 \end{cases} \quad \text{(triangle) (2)}$$

$$f(t) = (2t^3 - 3t^2 + 1)a_s + (-2t^3 + 3t^2)a + (t^3 - 2t^2 + t)g_s \quad \text{(Hermite function) (3)}$$

with time $t \in [0,1]$, the amplitude a_s at the beginning of the FAP at t=0, and the gradient g_s of $f(0)$ which is the FAP amplitude over time at t=0. If the transition time $T \neq 1$, the time axis of the functions (1) to (4) has to be scaled. These functions depend on a_s, g_s, a and T, and thus they are completely determined as soon as the FAP bookmark is known.

The Hermite function of third order enables one to match the tangent at the beginning of a segment with the tangent at the end of the previous segment, so that a smooth curve can be guarantied. Usually, the computation of the Hermite function requires 4 parameters as input, which are a_s, g_s, a and the gradient of $f(t)$ at t=1. Here a horizontal gradient at the end of the transition time is assumed. The gradient $g_s(t)$ at time t is computed according to

$$g_s(t) = (6t^2 - 6t)(a_s - a) + (3t^2 - 4t + 1)g_s \quad (4)$$

with a_s, g_s ,and a defined by the amplitude prior to the current bookmark.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex D
(normative)

Video buffering verifier

D.1 Introduction

The video verifier comprises the following models, as shown in Figure D.1:

1. A video rate buffer model
2. A video complexity model, and
3. A video reference memory model

A video rate buffer model is required in order to bound the memory requirements for the bitstream buffer needed by a video decoder. With a rate buffer model, the video encoder can be constrained to make bitstreams which are decodable with a predetermined buffer memory size.

A video complexity model is required in order to bound the processing speed requirements needed by a compliant video decoder. With a video complexity model, the video encoder can be constrained to make bitstreams which are decodable with a predetermined decoder processor capability.

A video reference memory model is required in order to bound the macroblock (pixel) memory requirements needed by a video decoder. With a video reference memory model, a video encoder can be constrained to make bitstreams which are decodable with a predetermined reference memory size.

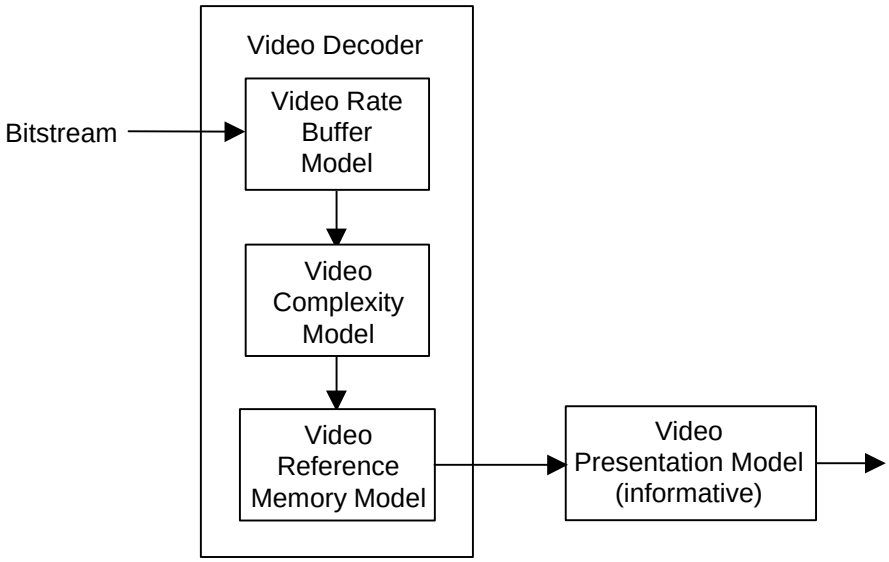


Figure D.1 — Video Verifier Model

D.2 Video Rate Buffer Model Definition

The ISO/IEC 14496-2 video buffering verifier (VBV) is an algorithm for checking a bitstream with its delivery rate function, R(t), to verify that the amount of rate buffer memory required in a decoder is less than the stated buffer size. If a visual bitstream is composed of multiple VOs each with one or more VOLs, the rate buffer model is applied independently to each VOL (using buffer size and rate functions particular to that VOL).

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The VBV applies to natural video coded as a combination of I, P, B and S-VOPs, and still texture. Face animation and mesh objects are not constrained by this model.

The coded video bitstream shall be constrained to comply with the requirements of the VBV defined as follows:

1. When the vbv_buffer_size and vbv_occupancy parameters are specified by systems-level configuration information, the bitstream shall be constrained according to the specified values. When the vbv_buffer_size and vbv_occupancy parameters are not specified (except in the short video header case as described below), this indicates that the bitstream should be constrained according to the default values of vbv_buffer_size and vbv_occupancy. The default value of vbv_buffer_size is the maximum value of vbv_buffer_size allowed within the profile and level. The default value of vbv_occupancy is 170 × vbv_buffer_size, where vbv_occupancy is in 64-bit units and vbv_buffer_size is in 16384-bit units. This corresponds to an initial occupancy of approximately two-thirds of the full buffer size.

2. The VBV buffer size is specified by the vbv_buffer_size field in the VOL header in units of 16384 bits. A vbv_buffer_size of 0 is forbidden. Define B = 16384 × vbv_buffer_size to be the VBV buffer size in bits.

3. The instantaneous video object layer channel bit rate seen by the encoder is denoted by $R_{vol}(t)$ in bits per second. If the bit_rate field in the VOL header is present, it defines a peak rate (in units of 400 bits per second; a value of 0 is forbidden) such that $R_{vol}(t)$ <= 400 × bit_rate. Note that $R_{vol}(t)$ counts only visual syntax for the current VOL (refer to the definition of d_i below). If the channel is a serial time mutiplex containing other VOLs or as defined by ISO/IEC 14496-1:2001 with a total instantaneous channel rate seen by the encoder of R(t), then

$$R_{vol}(t) = \begin{cases} R(t) & \text{if } t \in \{\text{channel bit duration of a bit from VOL } vol\} \\ 0 & \text{otherwise} \end{cases}$$

4. The VBV buffer is initially empty. The vbv_occupancy field specifies the initial occupancy of the VBV buffer in 64-bit units before decoding the initial VOP, subtracted by the length of any immediately preceding configuration information (see subclause 6.2.1). For basic sprite sequences, the vbv_occupancy field specifies the initial occupancy of the VBV buffer in 64-bit units before decoding the first S-VOP in the elementary stream (see subclause 6.2.1). The first bit in the VBV buffer is the first bit of the visual bitstream including the elementary stream and any immediately preceding configuration information for all types of sequences.

5. Define d_i to be size in bits of the i-th VOP plus any immediately preceding GOV header and configuration information and any immediately following stuffing_start_code and stuffing_byte(s), where i is the VOP index which increments by 1 in decoding order. A VOP includes any trailing stuffing code words before the next start code and the size of a coded VOP (d_i) is always a multiple of 8 bits due to start code alignment.

6. Let t_i be the decoding time associated with VOP i in decoding order. All bits (d_i) of VOP i are removed from the VBV buffer instantaneously at t_i. This instantaneous removal property distinguishes the VBV buffer model from a real rate buffer. The method of determining the value of t_i is defined in item 7 below.

7. τ_i is the composition time (or presentation time in a no-compositor decoder) of VOP *i*. For a video object plane, τ_i defined by vop_time_increment (in units of 1/vop_time_increment_resolution seconds) plus the cumulative number of whole seconds specified by modulo_time_base. In the case of interlaced video, a VOP consists of lines from two fields and τ_i is the composition time of the first field. The relationship between the composition time and the decoding time for a VOP is given by:

   $t_i = \tau_i$ if ((vop_coding_type of VOP *i* == B-VOP) || low_delay || scalability || sprite_enable == “static”)

   $t_i = \tau_{p(i)}$ otherwise

   Low_delay, scalability and sprite_enable are defined in subclause 6.3.3, and *p(i)* is the index of the nearest temporally-previous non-B VOP relative to VOP *i*. If low_delay, scalability and sprite_enable are all ‘0’, then the composition time of I, P, and S(GMC) VOP’s is delayed until all immediately temporally-previous B-VOPs have been composed; the decoding time of VOP *i* is then equal to the composition time of the temporally-previous non-B VOP instead of the composition time of VOP *i* itself.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For the first VOP (*i*=0*)*, if low_delay, scalability, and sprite_enable are all '0', it is necessary to define an initial decoding time t_0 (since the timing structure is locked to the B-VOP times and the first decoded VOP would not be a B-VOP). This defined decoding timing shall be that $t_0 = 2t_1 - t_2$ (i.e., assuming that $t_1 - t_0 = t_2 - t_1$), since $\tau_{p(0)}$ is not defined.

The following example demonstrates how t_i is determined for a sequence with variable numbers of consecutive B-VOPs:

Decoding order : $I_0 P_1 P_2 P_3 B_4 P_5 B_6 P_7 B_8 B_9 P_{10} B_{11} B_{12}$

Display order : $I_0 P_1 P_2 B_4 P_3 B_6 P_5 B_8 B_9 P_7 B_{11} B_{12} P_{10}$

Assume that vop_time_increment=1 and modulo_time_base=0 in this example. The sub-index *i* is in decoding order.

Table D.1 — An example that demonstrates how t_i is determined

i	τ_i	t_i	$p\ (i)$
0	0	-1	Not defined
1	1	0	0
2	2	1	1
3	4	2	2
4	3	3	Irrelevant
5	6	4	3
6	5	5	Irrelevant
7	9	6	5
8	7	7	Irrelevant
9	8	8	Irrelevant
10	12	9	7
11	10	10	Irrelevant
12	11	11	Irrelevant

8. Define b_i as the buffer occupancy in bits immediately following the removal of VOP i from the rate buffer. Using the above definitions, b_i can be iteratively defined

$$b_0 = 64 \times \text{vbv_occupancy} + (\text{size of configuration information}) - d_0$$

$$b_{i+1} = b_i + \int_{t_i}^{t_{i+1}} R_{vol}(t)dt - d_{i+1} \quad \text{for } i \geq 0$$

Since the first S-VOP in the elementary stream is regarded as the initial VOP (VOP 0) for basic sprite sequences in the above equation, the equation does not define the buffer occupancy for the I-VOP of basic sprite sequences. For defining the buffer occupancy, the I-VOP of a basic sprite sequence is divided into (N_{sp} + 395) / 396 *sections*, where N_{sp} is the number of macroblocks in the I-VOP. The *N*-th ($N \geq 0$) macroblock of the I-VOP (stuffing macroblocks are not included) in transmission order is included in the (*N* / 396)-th section. All the bits of the *k*-th ($0 \leq k < (N_{sp} + 395) / 396$) section are removed from the VBV buffer instantaneously at the section decoding time ts_k, where ts_k is the time when the last bit of this section enters the buffer. Thus the buffer occupancy immediately before the removal of section *k* is equal to the size in bits of this section, and the buffer occupancy immediately after the removal is 0. The initial section of the I-VOP includes all the bits from the beginning of the elementary stream to the end of the section. The last section includes all the stuffing bits added at the end of the I-VOP information.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9. The rate buffer model requires that the VBV buffer never overflow or underflow, that is

$$0 <= b_i \quad and \quad b_i + d_i <= B \quad for\ all\ i$$

For the initial I-VOP of basic sprite sequences, the VBV buffer occupancy immediately before the removal of a section shall not exceed *B*.

Real-valued arithmetic is used to compute b_i so that errors are not accumulated.

A coded VOP size must always be less than the VBV buffer size, i.e., $d_i < B$ for all i.

10. If the short video header is in use (i.e., when short_video_header = 1), then the parameter vbv_buffer_size is not present and the following conditions are required for VBV operation. The buffer is initially empty at the start of encoder operation (i.e., *t*=0 being at the time of the generation of the first video plane with short header), and its fullness is subsequently checked after each time interval of 1001/30000 seconds (i.e., at *t*=1001/30000, 2002/30000, etc.). If at least one complete VOP is in the buffer at the checking time, the data for the earliest VOP in bitstream order is removed. The buffer fullness after the removal of a VOP, b_i, shall be greater than or equal to zero and less than (4 *R*max 1001) / 30000 bits, where *R*max is the maximum bit rate in bits per second allowed within the profile and level. The number of bits used for coding any single VOP, d_i, shall not exceed *k* 16384 bits, where *k* = 4 for QCIF and Sub-QCIF, *k* = 16 for CIF, *k* = 32 for 4CIF, and *k* = 64 for 16CIF, unless a larger value of *k* is specified in the profile and level definition. Furthermore, the total buffer fullness at any time shall not exceed a value of *B* = *k* 16384 + (4 *R*max 1001) / 30000.

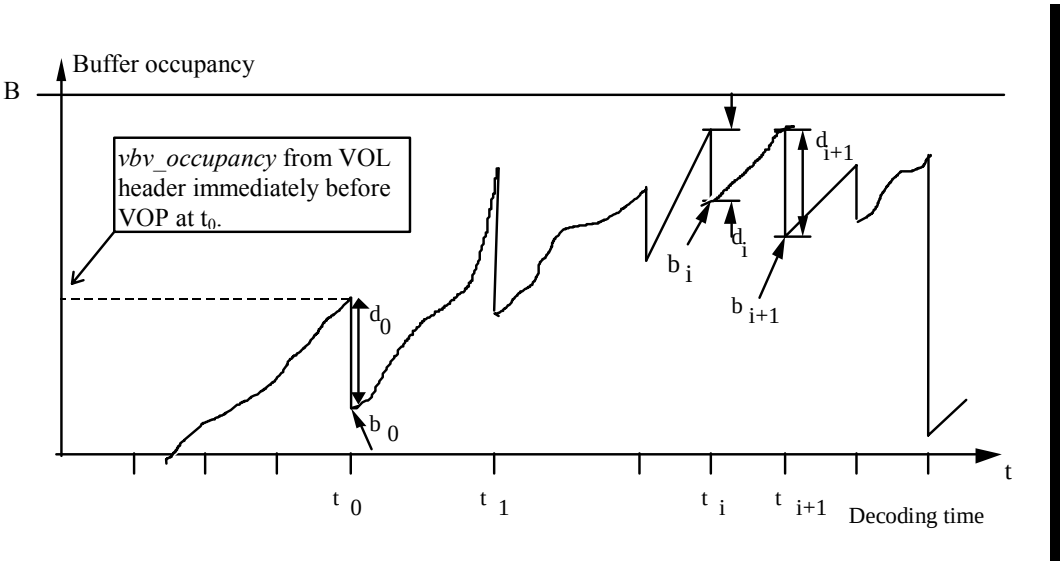


Figure D.2 — VBV Buffer occupancy

It is a requirement on the encoder to produce a bitstream which does not overflow or underflow the VBV buffer. This means the encoder must be designed to provide correct VBV operation for the range of values of $R_{vol,decoder}(t)$ over which the system will operate for delivery of the bitstream. A channel has constant delay if the encoder bitrate at time t when particular bit enters the channel, $R_{vol,encoder}(t)$ is equal to $R_{vol,decoder}(t + L)$, where the bit is received at (t + L) and L is constant. In the case of constant delay channels, the encoder can use its locally estimated $R_{vol,encoder}(t)$ to simulate the VBV occupancy and control the number of bits per VOP, d_i, in order to prevent overflow or underflow.

The VBV model assumes a constant delay channel. This allows the encoder to produce a bitstream which does not overflow or underflow the buffer using $R_{vol,encoder}(t)$ – note that $R_{vol}(t)$ is defined as $R_{vol,encoder}(t)$ in item 2 above.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

D.3 Comparison between ISO/IEC 14496-2 VBV and the ISO/IEC 13818-2 VBV (Informative)

The ISO/IEC 13818-2 and ISO/IEC 14496-2 VBV models both specify that the rate buffer may not overflow or underflow and that coded pictures (VOPs) are removed from the buffer instantaneously. In both models a coded picture/VOP is defined to include all higher-level syntax immediately preceding the picture/VOP regardless of whether the separate configuration/elementary streams mode or the combined configuration/elementary streams mode is used for the carriage of ISO/IEC 14496-2 bitstreams (see subclause 6.2.1). It is assumed that both the configuration information and the elementary stream is input to a common rate buffer memory.

ISO/IEC 13818-2:2000 has a constant frame period (although the bitstream can contain both frame and field pictures and frame pictures can use explicit 2:3 pulldown via the repeat_first_field flag). In ISO/IEC 14496 terms, this frame rate would be the output of the compositor (the ISO/IEC 13818 terminology is the output of the display process that is not defined normatively by ISO/IEC 13818). This output frame rate together with the ISO/IEC 13818-2 picture_structure and repeat_first_field flag precisely defines the time intervals between consecutive decoded picture (either frames or fields) passed between the decoding process and the display process.

ISO/IEC 13818-2 bitstream may contains B pictures (we assume ISO/IEC 13818-2 low_delay = 0). This means the coding order and display order of pictures is different (since both reference pictures used by a B picture must precede the B picture in coding order). The ISO/IEC 13818-2 VBV (and ISO/IEC 13818-1 systems T-STD) specifies that a B picture is decoded and presented (instantaneously) at the same time and the reference pictures are re-ordered to make this possible. This is the same reordering model specified in item 7 above.

A model ISO/IEC 14496-2 decoder using its VBV buffer model emulates a model decoder using the ISO/IEC 13838-2 VBV buffer model if the VOP time stamps given by vop_time_increment and the cumulative modulo_time_base agree with the sequence of ISO/IEC 13818-2 picture presentation times. We assume here that both coded picture/VOPs use the common subset of both standards (frame structured pictures and no 3:2 pulldown on the decoder, i.e., repeat_first_field = 0). For example, if the ISO/IEC 14496-2 sequence is coded at 29.97Hz (the NTSC picture rate), vop_time_increment_resolution will be 30000 and the change in vop_time_increment between consecutive VOPs in display order will be 1001 because picture skipping is not permitted in ISO/IEC 13818-2:2000 (when low_delay = 0).

D.4 Video Complexity Model Definition

The ISO/IEC 14496-2 video complexity verifier (VCV) is an algorithm for checking a bitstream to verify that the amount of processing capability required in a decoder is less than the stated complexity measure in macroblocks per second. This model applies to all macroblocks in a MPEG-4 visual bitstream.

A separate VCV buffer accumulates the boundary macroblocks of all VOLs in a combined MPEG-4 visual bitstream. The boundary MB VCV has a separate decoding rate (specified in Annex N) but it is subject to the same maximum value used in the VCV accumulating the total number of macroblocks. Boundary macroblocks (containing coded shape information which is not totally transparent or totally opaque) are included in both the VCV model and in the boundary VCV model. For S-VOPs in low-latency sprite sequences, the number of boundary macroblocks used in the boundary VCV model is defined as the number of boundary macroblocks with coded shape information included in sprite_shape_texture() (see subclauses 6.2.5.4 and 7.8.3).

The VCV applies to natural video coded as a combination of I, P, B and S-VOPs, while a separate VCV applies to still texture. Face animation and mesh objects are not constrained by this model. For basic and low-latency sprite sequences, an equivalent number of macroblocks is defined for each S-VOP.

The coded video bitstream shall be constrained to comply with the requirements of the VCV defined as follows:

1. A vcv_buffer_size is defined as the maximum number of MBs which can be contained in the VCV-buffer and is specified in Annex N. These MBs are consumed by the decoder at vcv_decoder_rate or H (in MB/s), as specified in Annex N for each profile and level. A vcv_decoder_latency or L is defined as the time needed for the decoder to decode a full VCV-buffer (vcv_buffer_size MBs) with the vcv_decoder_rate (MB/s). Thus the relation vcv_buffer_size = H*L holds. The VCV-buffer is initially empty at the start of decoding.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2. Let M_i be number of macroblocks in each VOP.

   For S-VOPs in basic or low-latency sprite sequences, M_i is $MB_{S\text{-}VOP}$. The hypothetical number of macroblocks in an S-VOP, $MB_{S\text{-}VOP}$, for the usage in the VCV model is defined as follows:

   $$MB_{S\text{-}VOP} = (MB_{rcn} > MB_{ref} \; ? \; MB_{rcn} : MB_{ref}) + MB_{lls},$$

   where MB_{rcn} is the number of macroblocks (including opaque, boundary, and transparent macroblocks) fully or partially included in the bounding rectangle of the reconstructed VOP, MB_{ref} is he number macroblocks included in the support area in the sprite, and MB_{lls} is the number of updated macroblocks in the sprite. MB_{ref} is defined as follows:

   $MB_{ref} = MB_{rcn}$, when no_of_sprite_warping_points == 0 or 1,

   $= (MB_{rcn} \times ((i_1'-i_0')^2 + (j_1'-j_0')^2))/(W^2 s^2)$, when no_of_sprite_warping_points == 2,

   $= (MB_{rcn} \times |(i_1' - i_0')(j_2' - j_0') - (i_2' - i_0')(j_1' - j_0')|) / (W H s^2)$, when no_of_sprite_warping_points == 3, and

   $= (MB_{rcn} \times (|(i_1' - i_0')(j_2' - j_0') - (i_2' - i_0')(j_1' - j_0')| + |(i_1' - i_3')(j_2' - j_3') - (i_2' - i_3')(j_1' - j_3')|))/(2 W H s^2)$, when no_of_sprite_warping_points == 4,

   where the definition of W, H, s, i_0', j_0', i_1', j_1', i_2', j_2', i_3', and j_3' is described in subclause 7.8.4. MB_{lls} denotes the number of macroblocks with at least one coded DCT coefficient included in sprite_shape_texture() (see subclauses 6.2.5.4 and 7.8.3). The value of MB_{lls} is 0 when low_latency_sprite_enable == 0.

   For the VOP with vop_reduced_resolution flag equal to 1, M_i is defined as the number of macroblocks in the $VOP_i \times 4$.

3. At time t_i, M_i is added to the VCV-buffer occupancy, v(t), where t_i is the VOP decoding time calculated according to subclause D.2 for each VOL. For the I-VOP in a basic sprite sequence, the number of macroblocks included in section *k* (see item 8 of subclause D.2 for the definition of a section), MS_k, is added to the VCV buffer occupancy at ts_k, where ts_k is the section decoding time defined in subclause D.2. The occupancy of the VCV buffer decreases linearly at rate H (defined in Table N.1) until the occupancy is zero or until the time reaches t_{next} where t_{next} is the earliest VOP or section decoding time greater than t_i or ts_k for some VOP in the visual bitstream. If the VCV occupancy becomes zero (the VCV decoder is idle), then the VCV remains idle until t_{next}. See Figure D.3.

4. The interval of time where VOP i is being decoded extends from s_i to e_i which are defined as

   $$s_i = t_i + \text{Max}[v(t_i)/H, v_B(t_i)/H_B]$$
   $$e_i = t_i + \text{Max}[(v(t_i) + M_i)/H, (v_B(t_i) + M_{Bi})/H_B]$$

   $v(t_i)$ is the VCV occupancy before the macroblocks representing VOP i, M_i, are added to v(t),
   H is the VCV decoder rate (see Table N.1),
   $v_B(t_i)$ is the boundary MB VCV occupancy before the boundary macroblocks representing VOP i, M_{Bi}, are added to $v_B(t)$, and
   H_B is the VCV boundary MB decoder rate (see Table N.1).

   For the I-VOP in a basic sprite sequence, the interval of time where section *k* is being decoded extends from ss_k to es_k which are defined as

   $$ss_k = ts_k + \text{Max}[v(ts_k)/H, v_B(ts_k)/H_B]$$
   $$es_k = ts_k + \text{Max}[(v(ts_k) + MS_k)/H, (v_B(ts_k) + MS_{Bk})/H_B]$$

   $v(ts_k)$ is the VCV occupancy before the number of macroblocks in section *k*, MS_k, is added to the VCV occupancy, and
   $v_B(ts_k)$ is the boundary MB VCV occupancy before the boundary macroblocks representing section *k*, MS_{Bk}, are added to $v_B(ts_k)$.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5. To decode a VOL, the decoding of each VOP i must be complete by τ_i + L (composition time plus the latency of the VCV decoding process). The latency L of the VCV decoding process is constant for all VOPs. This constraint does not apply to the I-VOP in a basic sprite sequence.

6. The complexity model allows the VCV buffer to underflow, in which case the decoder remains idle, and the VCV buffer occupancy, v(t), remains during the idle period.

D.5 Video Reference Memory Model Definition

The ISO/IEC 14496-2 video memory verifier (VMV) is an algorithm for checking a bitstream to verify that the amount of pixel memory required in a decoder is less than the maximum VMV buffer size (in units of MBs) specified in Annex N for each profile and level. If a visual bitstream is composed of multiple VOs each with one or more VOLs, the VMV models the memory requirements of all VOLs (this model assumes a shared memory space shared by all VOLs of all VOs).

The VMV applies to natural video coded as a combination of I, P, B, S-VOPs, and still texture. Face animation, mesh objects and I-VOPs in basic sprite sequences are not constrained by this model.

The coded video bitstream shall be constrained to comply with the requirements of the VMV defined as follows:

1. The reference memory is initially empty. It is filled with the decoded data as each macroblock is decoded.

2. For I, P, B, and S(GMC)-VOPs, the amount of reference memory required for the decoding of the ith VOP is defined as the number of macroblocks in the VOP, M_i. For the VOP with vop_reduced_resolution flag equal to 1, M_i is defined as the number of macroblocks in the $VOP_i \times 4$. This memory is consumed at the same constant rate of

$$\frac{M_i}{e_i - s_i} \quad \text{(MB/s)}$$

   as the decoding process occurs.

   For S-VOPs in basic or low-latency sprite sequences, the amount of reference memory required for the decoding of the VOP is defined as the number of macroblocks in the reconstructed VOP, MB_{rcn} (see item 2 of subclause D.4 for the definition of MB_{rcn}). This memory is consumed at the constant rate of

$$\frac{MB_{rcn}}{e_i - s_i} \quad \text{(MB/s)}$$

   as the decoding process occurs. The memory used for storing the sprite is ignored by the VMV.

   The decoding duration of VOP i, T_i , is identical in the VCV and VMV models and starts at s_i and ends at e_i as defined in subclause D.4 above.

3. At the composition time (or presentation time in a no-compositor decoder) plus VCV latency, τ_i + L, of an I, P, or S(GMC)-VOP the total memory allocated to the previous I, P, or S(GMC)-VOP in the decoding order is released instantaneously.

4. At the composition time (or presentation time in a no-compositor decoder) plus VCV latency, τ_i + L, of a B-VOP, or an S-VOP in a basic or low latency sprite sequence, the total memory allocated to that B-VOP or S-VOP is released instantaneously.

5. In case of scalability coding, the release of VMV buffer is managed by each VOL independently. In addition to this constraint, the time of memory release for any VOP in enhancement layer follows definition 6 below.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6. At the composition time (or presentation time in a no-compositor decoder) plus VCV latency, τ_i +L, of a I-, P- or B-VOP of enhancement layer, the total memory allocated to the previous I-, P- or B-VOP in the decoding order is released instantaneously.

7. The reference memory model requires that the VMV buffer never exceeds the vmv_buffer_size obtained from the Table N.1. The buffer occupancy of the video memory verifier is shown in Figure D.3.

D.6 Interaction between VBV, VCV and VMV (informative)

The VBV model defines when the bitstream is available for the decoding and is removed from the buffer. The VCV model defines the speed at which the macroblocks are decoded. The reference memory model defines the amount of reference memory that is consumed and released. Obviously, it is advantageous for the video decoder to decode as far in advance as possible. However, this is constrained by the VBV and the VMV. The decoder can only start decoding if the bits are available for decoding. At the same time as the decoder decodes the bitstream, it generates macroblocks which consumes the reference memory. So if it decodes too fast it will overflow the reference memory.

On the other hand if the decoder start decoding too late, then it will not be able to complete the decoding in time and the bitstream will be removed from the VBV before it could be processed. Similarly the reference memory required for the prediction of the current VOP may also be removed from the VMV.

Therefore, the encoder will have to simulate the VBV, VCV and VMV in order to produce a bitstream compliant with the intended profile and level. If the simulated decoder VBV becomes too full, the encoder should produce more bits (decrease the coarseness of quantisation) or generate stuffing bits to prevent overflow. If the VBV occupancy becomes low, fewer bits are to produced. If the VCV approaches its limit, the encoder must allow more time for the decoder to produce its output. This can be done, for example, by increasing the interval of time between VOPs If the VMV becomes too full, decoder memory usage must be decreased. Using smaller VOPs (fewer macroblocks) and avoiding B-VOPs reduce the decoder's memory requirements.

D.7 Video Presentation Model Definition (informative)

The video presentation verifier (VPV) is not part of this specification. It is an algorithm for checking a bitstream to verify that the amount of presentation buffer required in a decoder is less than a given amount of memory in units of MB. It is also used to constraint the speed of the compositor in terms of maximum number of MB/sec.

The VPV operates in the same manner as the VCV:

1. At the composition time plus VCV decoder latency of the ith VOP, τ_i, + L, the VOP is placed in the presentation buffer.

2. The data in the presentation buffer is consumed by the compositor at a rate equivalent to the maximum number of MB/sec.

3. At τ_i + L + compositor_latency the VOP should be composited.

4. The presentation memory model requires that the VPV buffer never overflows

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

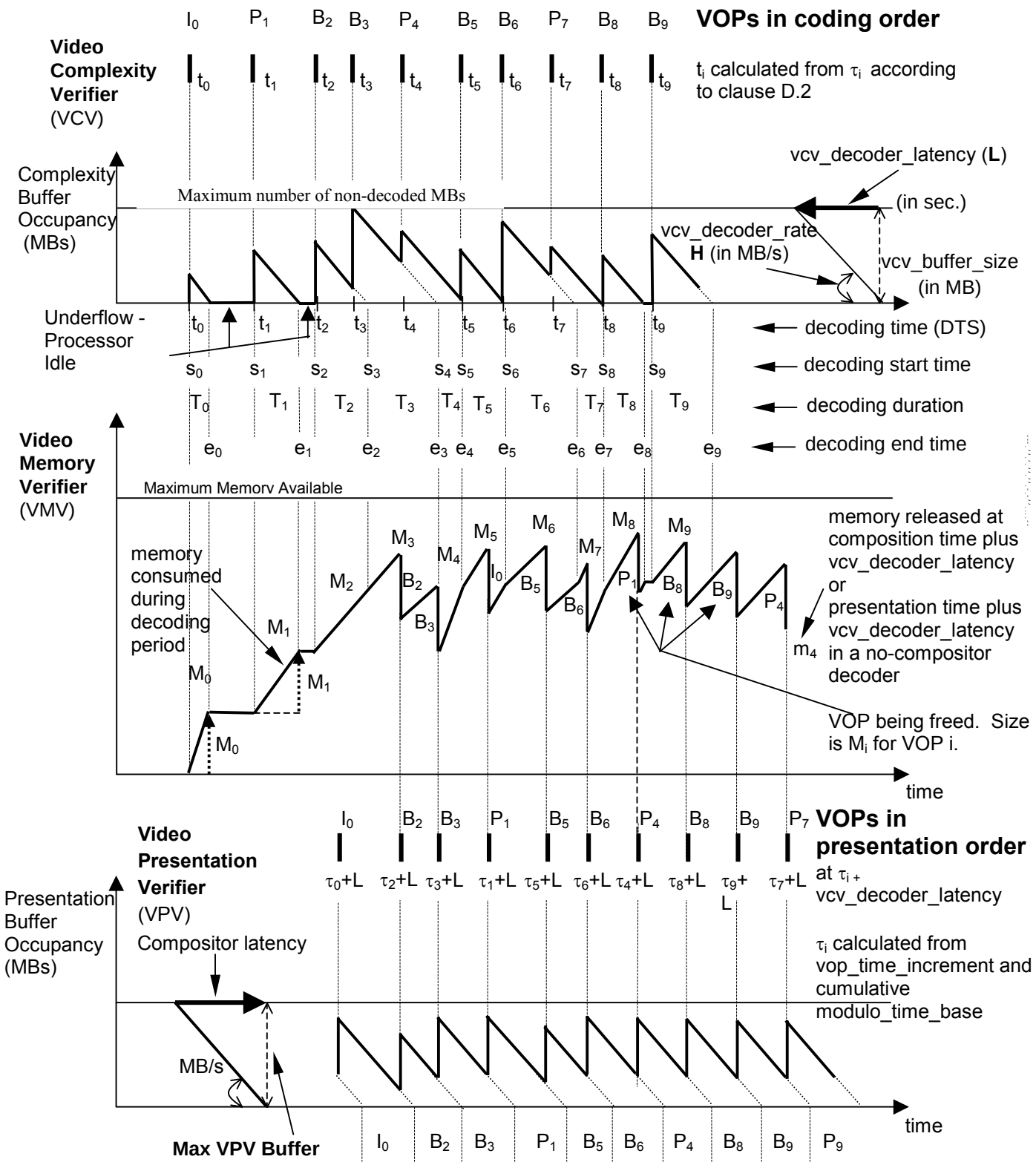


Figure D.3 — VCV, VMV and VPV buffer occupancy (video_object_layer_shape == "rectangular")

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Annex E
(informative)

Features supported by the algorithm

E.1 Error resilience

E.1.1 Resynchronization

Resynchronization tools, as the name implies, attempt to enable resynchronization between the decoder and the bitstream after a residual error or errors have been detected. Generally, the data between the synchronization point prior to the error and the first point where synchronization is reestablished, is discarded. If the resynchronization approach is effective at localizing the amount of data discarded by the decoder, then the ability of other types of tools that recover data and/or conceal the effects of errors is greatly enhanced.

The resynchronization approach adopted by ISO/IEC 14496, referred to as a packet approach, is similar to the Group of Blocks (GOBs) structure utilized by the ITU-T Recommendations H.261 and H.263. In these standards a GOB is defined as one or more rows of macroblocks (MB). At the start of a new GOB, information called a GOB header is placed within the bitstream. This header information contains a GOB start code, which is different from a picture start code, and allows the decoder to locate this GOB. Furthermore, the GOB header contains information which allows the decoding process to be restarted (i.e., resynchronize the decoder to the bitstream and reset all coded data that is predicted).

The GOB approach to resynchronization is based on spatial resynchronization. That is, once a particular macroblock location is reached in the encoding process, a resynchronization marker is inserted into the bitstream. A potential problem with this approach is that since the encoding process is variable rate, these resynchronization markers will most likely be unevenly spaced throughout the bitstream. Therefore, certain portions of the scene, such as high motion areas, will be more susceptible to errors, which will also be more difficult to conceal.

The video packet approach adopted by ISO/IEC 14496, is based on providing periodic resynchronization markers throughout the bitstream. In other words, the length of the video packets are not based on the number of macroblocks, but instead on the number of bits contained in that packet. If the number of bits contained in the current video packet exceeds a predetermined threshold, then a new video packet is created at the start of the next macroblock.

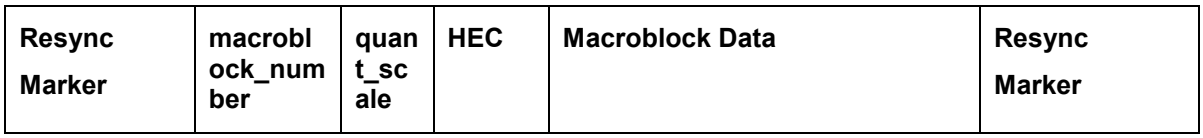


Figure E.1 — Error Resilient Video Packet

In Figure E.1, a typical video packet is described. A resynchronization marker is used to distinguish the start of a new video packet. This marker is distinguishable from all possible VLC code words as well as the VOP start code. Header information is also provided at the start of a video packet. Contained in this header is the information necessary to restart the decoding process and includes: the macroblock address (number) of the first macroblock contained in this packet and the quantiser scale (quant_scale) necessary to decode that first macroblock. The macroblock number provides the necessary spatial resynchronization while the quantiser scale allows the differential decoding process to be resynchronized. Following the quant_scale is the Header Extension Code (HEC). As the name implies, HEC is a single bit used to indicate whether additional information will be available in this header. If the HEC is equal to one then the following additional information is available in this packet header: modulo_time_base, vop_time_increment, vop_coding_type, intra_dc_vlc_thr, vop_fcode_forward, vop_fcode_backward.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The Header Extension Code makes each video packet (VP) possible to be decoded independently, when its value is equal to 1. The necessary information to decode the VP is included in the header extension code field, if the HEC is equal to 1.

If the VOP header information is corrupted by the transmission error, they can be corrected by the HEC information. The decoder can detect the error in the VOP header, if the decoded information is inconsistent with its semantics. For example, because it is prohibited that the values of the vop_fcode_forward and vop_fcode_backward are set to "0", if they are 0, the decoder can detect the error in the fcode information. In such a case, the decoder can correct the value by using the HEC information of the next VP.

When utilizing the error resilience tools within ISO/IEC 14496, some of the compression efficiency tools are modified. For example, all predictively encoded information must be confined within a video packet so as to prevent the propagation of errors. In other words, when predicting (i.e., AC/DC prediction and motion vector prediction) a video packet boundary is treated like a VOP boundary.

In conjunction with the video packet approach to resynchronization, a second method called fixed interval synchronization has also been adopted by ISO/IEC 14496. This method requires that VOP start codes and resynchronization markers (i.e., the start of a video packet) appear only at legal fixed interval locations in the bitstream. This helps to avoid the problems associated with start codes emulations. That is, when errors are present in a bitstream it is possible for these errors to emulate a VOP start code. In this case, when fixed interval synchronization is utilized the decoder is only required to search for a VOP start code at the beginning of each fixed interval. The fixed interval synchronization method extends this approach to be any predetermined interval.

The fixed interval synchronization is achieved by first inserting a bit with the value 0 and then, if necessary, inserting bits with the value 1 before the start code and the resync marker. The video decoder can determine if errors are injured in a video packet by detecting the incorrect number of the stuffing bits. (e.g. eight or more 1's are followed after 0 at the last part of a video packet, or the remaining bit pattern is not "011...")

E.1.2 Data Partitioning

Error concealment is an extremely important component of any error robust video codec. Similar to the error resilience tools discussed above, the effectiveness of an error concealment strategy is highly dependent on the performance of the resynchronization scheme. Basically, if the resynchronization method can effectively localize the error then the error concealment problem becomes much more tractable. For low bitrate, low delay applications the current resynchronization scheme provides very acceptable results with a simple concealment strategy, such as copying blocks from the previous frame.

In recognizing the need to provide enhanced concealment capabilities, the Video Group has developed an additional error resilient mode that further improves the ability of the decoder to localize an error. Specifically, this approach utilizes data partitioning. This data partitioning is achieved by separating the motion and macroblock header information away from the texture information. This approach requires that a second resynchronization marker be inserted between motion and texture information. Data partitioning, like the use of RVLCs, is signaled to the decoder in the VOL. Figure E-2 illustrates the syntactic structure of the data partitioning mode. If the texture information is lost, this approach utilizes the motion information to conceal these errors. That is, due to the errors the texture information is discarded, while the motion is used to motion compensate the previously decoded VOP.

Resync Marker	macroblock_number	quant_scale	HEC	Motion &Header Information	Motion Marker	Texture Information	Resync Marker

Figure E.2 — Data Partitioning

E.1.3 Reversible VLC

Reversible Variable Length Codes (RVLC) are designed such that they can be instantaneously decoded both in forward and reverse directions. A part of a bitstream which cannot be decoded in the forward direction due to the presence of errors can often be decoded in the backward direction. This is illustrated in Figure E.3. Therefore number of discarded bits can be reduced. RVLC is applied only to Tcoef coding

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

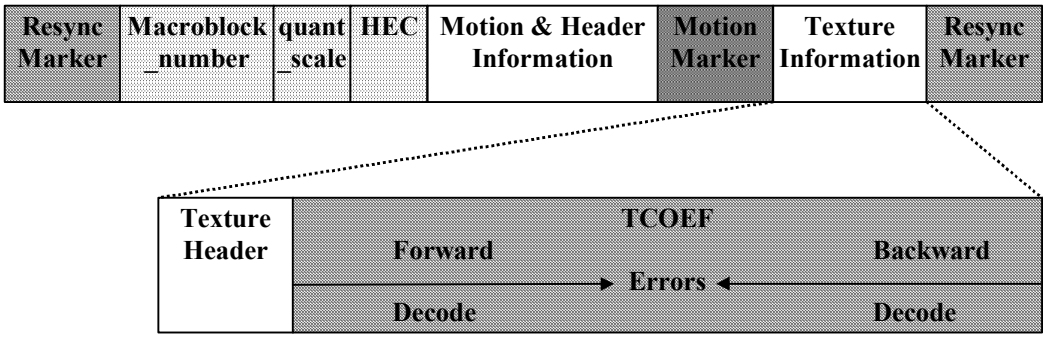


Figure E.3 — Reversible VLC

E.1.4 Decoder Operation

E.1.4.1 General Error Detection

An illegal VLC is received.

A semantic error is detected.

More than 64 DCT coefficients are decoded in a block.

Inconsistent resyncronization header information (i.e., QP out of range, MBA(k)<MBA(k-1),etc.)

E.1.4.2 Resynchronization

When an error is detected in the bitstream, the decoder should resynchronize at the next suitable resynchronization point(vop_start_code or resync_marker).

After that, it can be determined by detecting the incorrect number of the stuffing bits whether or not errors are injured in a video packet. If eight or more 1's are followed after 0 at the last part of a video packet, or the remaining bit pattern is not "011…", it means there is any error in this video packet.

If the VOP start code is corrupted by the transmission error and the frame synchronization is lost, the decoder may establish the resynchronization by using the HEC information. The decoder compares the vop_time_increment in the VOP header with one in the HEC field. If they are not same, the decoder may find that the current VOP start code is corrupted by the error. In this case, there must not be the error in the both vop_time_increments. The simple method is to check whether the vop_time_increment is mutilple of frame interval of the original source format (NTSC, PAL and so on). Therefore, it is expected that the number of the vop_time_increment is as many as possible. As this check method does not always detect the error, this is the auxiliary technique.

Missing blocks may be replaced with the same block from the previous frame.

E.1.4.3 Data Partitioning

E.1.4.4 Reversible VLC

This subclause describes a decoding methodology for Reversible Variable Length Codes (RVLC) when errors in the video bitstream are detected during the decoding process. This particular decoding methodology was developed during the RVLC core experiment process.

E.1.4.4.1 Process for detecting errors for both forward and backward decoding

Errors are present in the following cases:

(1) An illegal RVLC is found, where an illegal RVLC is defined as follows:

A codeword whose pattern is not listed in the RVLC table (e.g. 169 codeword patterns and escape codes).

Escape coding is used (i.e., a legal codeword is not available in the RVLC table) and the decoded value for LEVEL is zero.

--`,,`-`-`,,`,,`,`,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The second escape code is incorrect (e.g. codeword is not “00000” or “00001” for forward decoding, and/or is not “00001” for backward decoding).

There is a decoded value of FLC part using escape codes (e.g. LAST, RUN, LEVEL) in the RVLC table.

An incorrect number of stuffing bits for byte alignment (e.g. eight or more 1s follow 0 at the last part of a Video packet (VP), or the remaining bit pattern is not “0111...” after decoding process is finished).

(2) More than 64 DCT coefficients are decoded in a block.

E.1.4.4.2 Decoding information

The bitstream is decoded in the forward direction first. If no errors are detected, the bitstream is assumed to be valid and the decoding process is finished for that video packet. If an error is detected however, two-way decoding is applied. The following strategies for determining which bits to discard are used. These strategies are described using the figures given below along with the following definitions:

L : Total number of bits for DCT coefficients part in a VP.
N : Total number of macroblocks (MBs) in a VP.
L1 : Number of bits which can be decoded in a forward decoding.
L2 : Number of bits which can be decoded in a backward decoding.
N1 : Number of MBs which can be completely decoded in a forward decoding.
(0 <= N1 <= (N-1))
N2 : Number of MBs which can be completely decoded in a backward decoding.
(0 <= N2 <= (N-1))
f_mb(S) : Number of decoded MBs when S bits can be decoded in a forward direction.
(Equal to or more than one bit can be decoded in a MB, f_mb(S) counter is up.)
b_mb(S) : Number of decoded MBs when S bits can be decoded in a backward direction.
T : Threshold (90 is used now).

E.1.4.4.2.1 Strategies for decoding RVLC

(1) Strategy 1 : *L1+L2 < L* and *N1+N2 < N*

MBs of *f_mb(L1-T)* from the beginning and MBs of *b_mb(L2-T)* from the end are used. In the figure below, the MBs of the dark part are discarded.

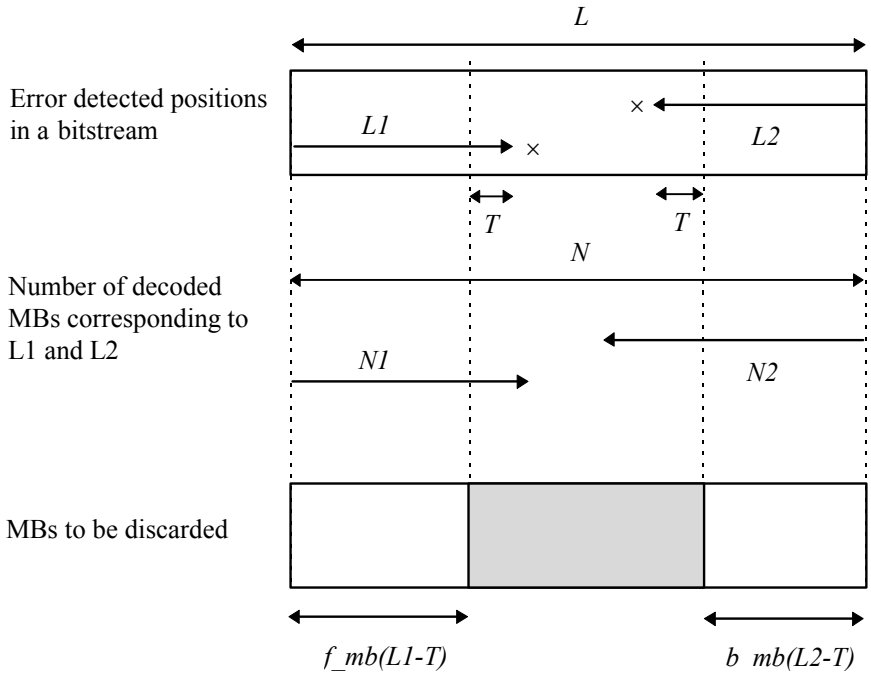


Figure E.4 — RVLC decoding; Strategy 1

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

(2) Strategy 2 : *L1+L2 < L* and *N1+N2 >= N*

MBs of *N-N2-1* from the beginning and MBs of *N-N1-1* from the end are used. MBs of the dark part are discarded.

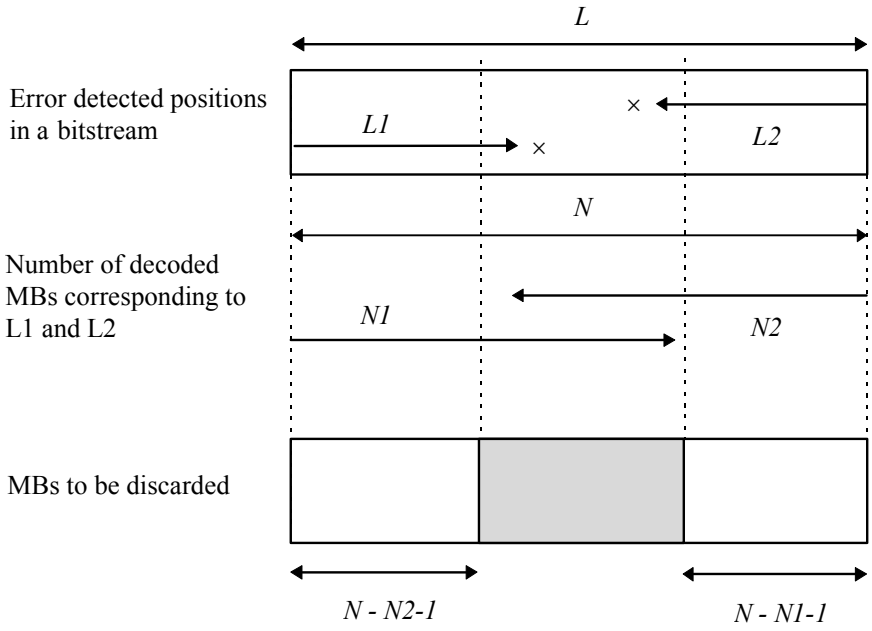


Figure E.5 — RVLC decoding; Strategy 2

(3) Strategy 3 : *L1+L2 >= L* and *N1+N2 < N*

MBs of *N-b_mb(L2)* from the beginning and MBs of *N-f_mb(L1)* from the end are used. MBs of the dark part are discarded.

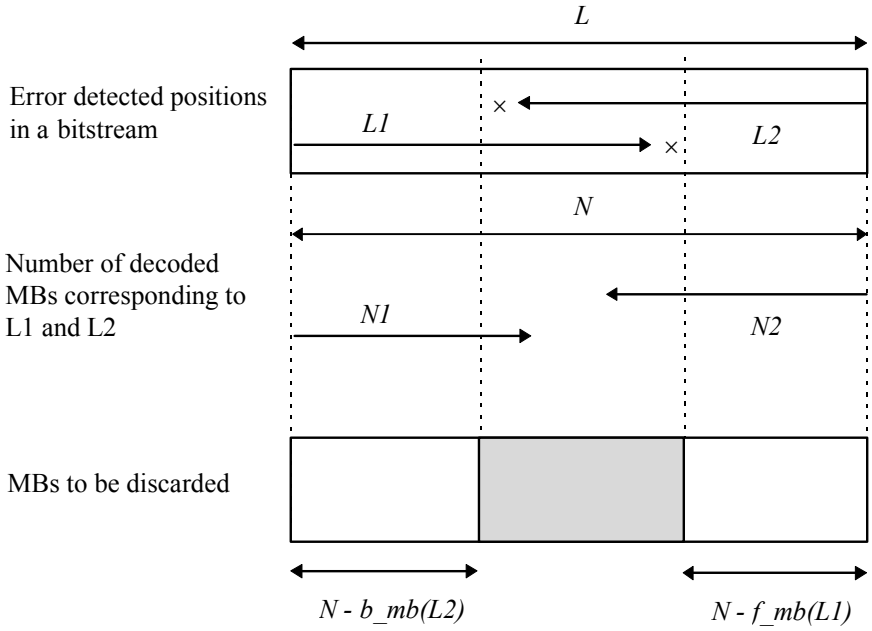


Figure E.6 — RVLC decoding; Strategy 3

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(4) Strategy 4 : *L1+L2* >= *L* and *N1+N2* >= *N*

MBs of *min{N-b_mb(L2), N-N2-1}* from the beginning and MBs of *min{N-f_mb(L1), N-N1-1}* from the end are used. MBs of the dark part are discarded.

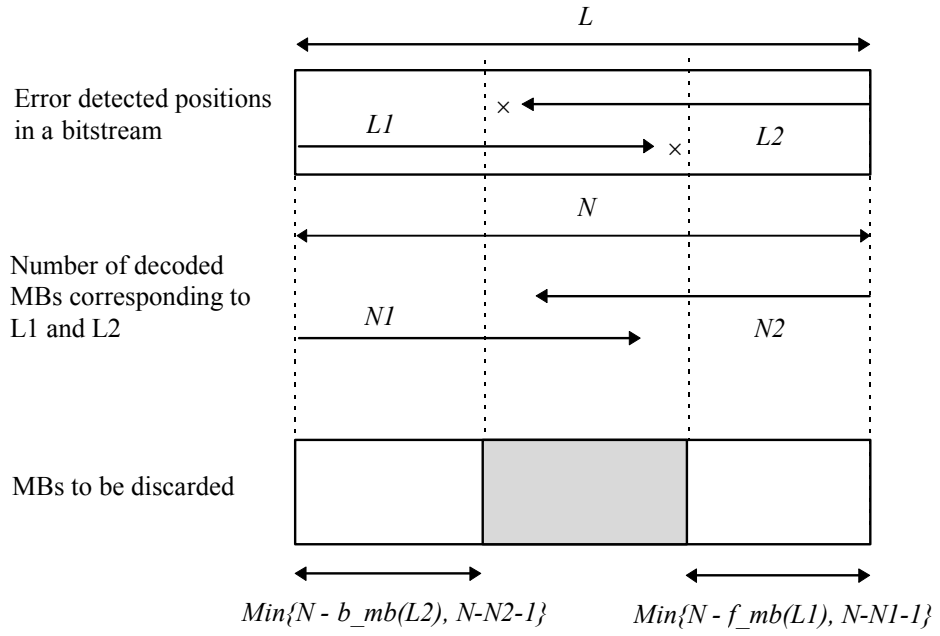


Figure E.7 — RVLC decoding; Strategy 4

E.1.4.4.2.2 INTRA MBs within a bitstream

In the above strategies (**Strategy 1** - **Strategy 4**), INTRA MBs are discarded even though they could have been decoded. An example of such a case is shown below.

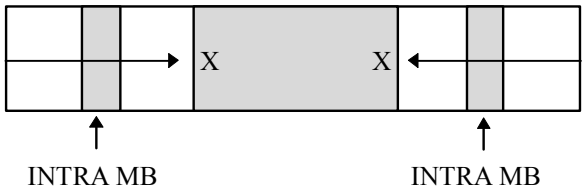


Figure E.8 — Intra MB discarding

Although these intra MBs are thought to be correct, the result of displaying an Intra MB that does contain an error can substantially degrade the quality of the video. Therefore, when a video packet is determined to contain errors, all Intra MBs are not displayed, but instead concealed.

E.1.5 Adaptive Intra Refresh (AIR) Method

The AIR is the technique of the intra refresh method for the error resilience. In the AIR, motion area is encoded frequently in Intra mode. Therefore, it is possible to recover the corrupted motion area quickly.

NOTE This informative technique is used for the rectangular VOP. It was designed in order to extract the motion area from the rectangular VOP at the encoder and encode it in intra mode frequently. Therefore, the quality of this area that is corrupted by the transmission error can be recovered quickly. As the objects are extracted in advance in the arbitrary shape coding mode, it is possible to obtain the same performance by using the conventional cyclic intra refresh within the arbitrary shape.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The method of the "AIR"

The number of Intra MBs in a VOP is fixed and pre-determined. It depends on bitrates and frame rate and so on.

The encoder estimates motion of each MB and the only motion area is encoded in Intra mode. The results of the estimation are recorded to the **Refresh Map** MB by MB. The encoder refers to the Refresh Map and decides to encode current MB in Intra mode or not. The estimation of motion is performed by the comparison between SAD and SAD_th. SAD is the Sum of the Absolute Differential value between the current MB and the MB in same location of the previous VOP. The SAD has been already calculated in the Motion Estimation part. Therefore, additional calculation for the AIR is not needed. SAD_th is the threshold value. If the SAD of the current MB is larger than the SAD_th, this MB is regarded as motion area. Once the MB is regarded as motion area, it is regarded as motion area until it is encoded in Intra mode predetermined times. The predetermined value is recorded to the Refresh Map. (*See* Figure E.9. *In this figure, predetermined value is "1" as an example*)

The holizontal scan is used to determine the MBs to be encoded in Intra mode within the moving area (*see* Figure E.10).

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	0	1	0	0
0	0	0	1	1	1	1	1	1	1	0
0	0	0	0	0	1	1	1	1	1	0
0	0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure E.9 — Refresh Map for QCIF

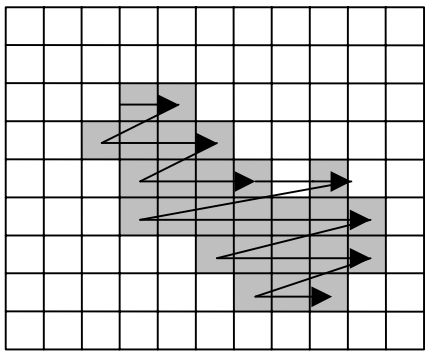


Figure E.10 — Scan order for the Adaptive Refresh

The processing of the "AIR"

The following is the explanation of the processing of AIR (see Figure E.11). The fixed number of the Intra MB in a VOP should be determined in advance. Here, it is set to "2" as an example.

[1] 1st VOP ([a]~[b] in Figure E.11)
The all MBs in the 1st VOP are encoded in Intra mode [a]. The Refresh Map is set to "0", because there is no previous VOP [b].

[2] 2nd VOP ([c] ~ [f])
The 2nd VOP is encoded as P-VOP. Intra refresh is not performed in this VOP, because all values in the Refresh Map is zero yet ([c] and [d]). The encoder estimates motion of each MB. If the SAD for current MB is larger than the SAD_th, it is regarded as motion area (hatched area in Figure E.11 [e]). And the Refresh Map is updated [f].

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[3] 3rd VOP ([g] ~ [k])
When the 3rd VOP is encoded, the encoder refers to the Refresh Map [g]. If the current MB is the target of the Intra refresh, it is encoded in Intra mode [h]. The value of the MB in Refresh Map is decreased by 1 [i]. If the decreased value is 0, this MB is not regarded as motion area. After this, the processing is as same as the 2nd VOP [j]~[k].

[4] 4th VOP ([l]~[p])
It is as same as 3rd VOP

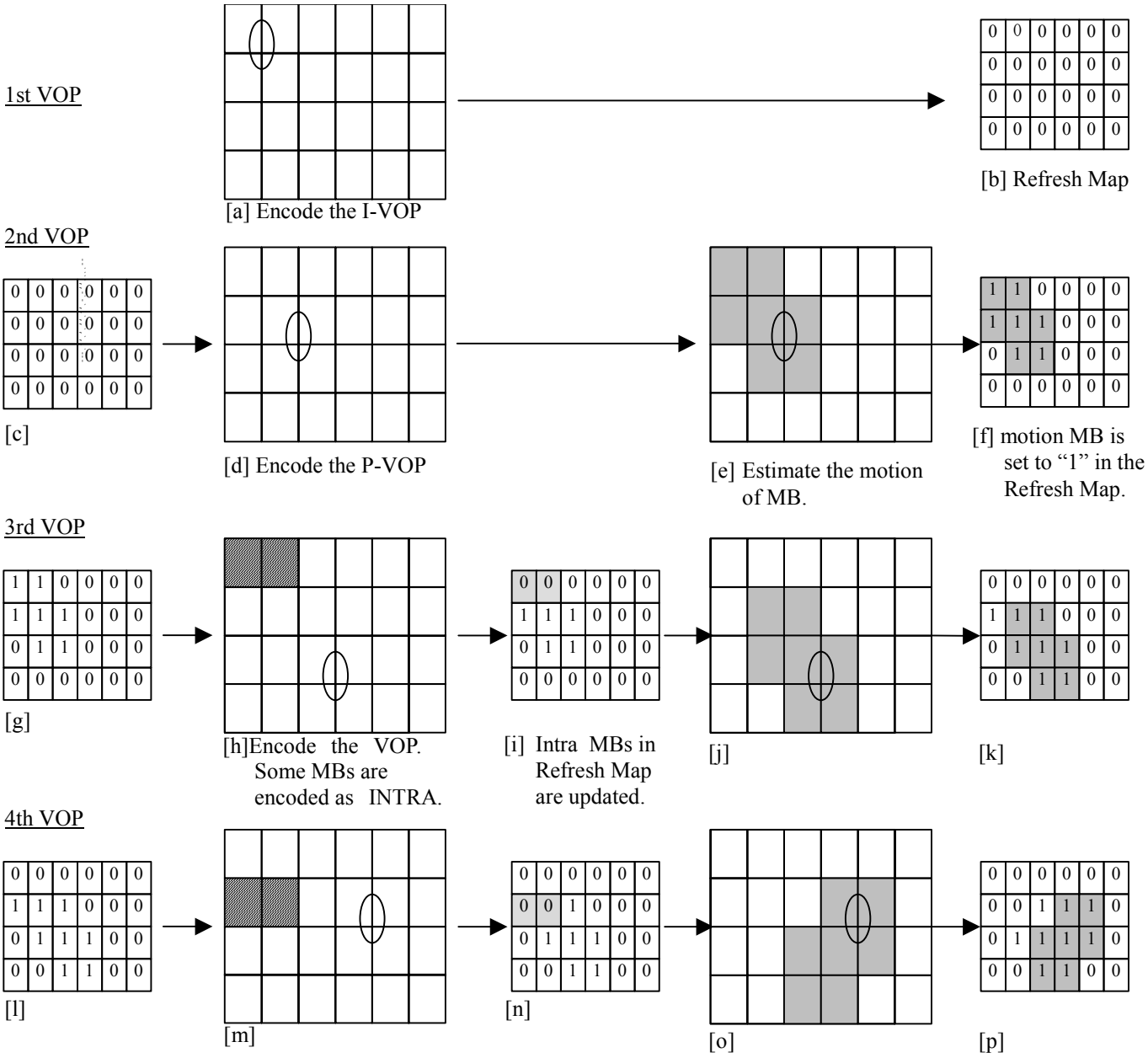


Figure E.11 — Explanation of AIR

E.1.6 NEWPRED

Recovery from temporal error propagation is an indispensable component of any error robust video communication system. Errors introduced during transmission will lead to frame-store mismatch between the encoder and the decoder, which will continue until some intra refresh occurs. When an upstream does not exist, cyclic intra refresh or adaptive intra refresh are available. In the case where the upstream exists from the decoder to the encoder, NEWPRED or demand intra refresh can be used. This subclause is presenting decoding process when newpred_enable = '1'.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

E.1.6.1 Introduction to NEWPRED mode

NEWPRED mode is one of the error resilience tools. It uses upstream messages to indicate which NEWPRED segments (NP segment) are correctly decoded and which NP segments are erroneously decoded. The encoder, which receives an upstream message, uses only correctly decoded part for prediction in an inter-frame coding. This prevents a temporal error propagation without the insertion of Intra coded MBs and improves the picture quality in the erroneous environments.

E.1.6.2 Upstream channel

A special logical channel is opened for the upstream from the decoder to the encoder. The syntax of the upstream message is described in subclause 6.2.12.

It requires an external framing mechanism for synchronization of the messages within the upstream, as the back-channel syntax defined herein contains no synchronization flag words. This synchronization is provided by the lower layer such as ISO/IEC 14496-1:2001 (MPEG-4 Systems).

E.1.6.3 Definition of NEWPRED

NEWPRED is a technique in which the reference picture for inter-frame coding is replaced adaptively according to the upstream messaging from the decoder. One VOP or one Video Packet is treated as one NP segment. NP segment is indicated in newpred_segment_type in VOL header. The reference picture is replaced independently in each NP segment. When Video Packet is NP segment, any shape of NP segment shall be restricted to be rectangular.

The encoder has some kinds of mode for selection of the reference picture, and the strategy used by the encoder to select the reference picture is beyond the scope of this standard.

The decoder, however, takes the same action regardless of the modes. The decoder uses the NP segment which is indicated in the vop_id_for_prediction field as a reference picture if it exists in the picture buffer memory, and returns ether NP_ACK or NP_NACK message. This clause describes the processes of the decoder.

E.1.6.4 Decoder definition including upstream signaling

The decoder decodes a NP segment with reference to the previously decoded segment whose vop_id is vop_id_for_prediction as a referenced segment for inter-frame decoding. If vop_id_for_prediction_indication is ‘0’, the decoder refers to the immediately previous decoded segment, or it refers to no segment. It shall not, however, refer to any segment which is decoded before the last I_VOP.

The decoder returns the upstream message to the encoder based on the upstream syntax defined in 7.14.2. The requested_upstream_type in VOL layer indicates which type of upstream message is needed by the encoder. The decoder returns only requested type of upstream message. The decoder, however, does not have to return any upstream message for B-VOP, because B-VOP is never referred by other VOP after that. After the received NP segment is correctly decoded, the ACK message of the NP segment is returned. When any error is detected during the decoding process or the error indication is delivered from the lower system layer, the decoder returns a NP_NACK message with the requested_vop_id_for_prediction which is the vop_id of the last correctly decoded NP segment of the same location of the current NP segment in the buffer memory. The error event includes the case that the NP segment whose vop_id is vop_id_for_prediction does not exist in the decoder. This occurs when the NP segment cannot be successfully decoded due to the previous errors. The vop_id field of the upstream message is copied from the corresponding field of the downstream message when the reliable vop_id is available. Otherwise, it may happen in the case of NP_NACK, vop_id is incremented by 1 from the reliable vop_id of the previously received NP segment of the same location of the current NP segment.

The decoder adds the correctly decoded NP segment into the buffer memory. If any errors are detected in the decoding process or the error indication is delivered from the lower system layer, the NP segment may not be added into the buffer memory. In the case of the buffer memory overflow, the oldest NP segment of the same location except for the segment which is referenced in the preceding encoding, may be removed.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

E.1.6.5 Additional buffer memory for reference picture

When the NP_ACK message is requested by the encoder, the decoder may need several number of buffer memory for reference picture. The needed number of the buffer memories is according to the strategy of the reference picture selection by the encoder, and transmission delay between the encoder and the decoder. If the encoder selects the reference picture according NP_NACK message, the additional memory is not always needed. When some error concealment techniques are used in the decoder, it is desirable that the decoder has at least one additional memory.

E.1.6.6 Interaction between the video layer and the system layer

In the downstream, one NP segment is suggested to form a transmission unit of the system standard.

In the upstream, an upstream message is suggested to form a transmission unit of the system standard.

If the system layer detects the transmission errors in the downstream, it is desirable to deliver the error indication with video data in the decoder. In the upstream, the errored upstream message may be discarded in the encoder.

E.2 Complexity Estimation

The Complexity Estimation Tool enables the estimation of decoding complexity without the need of the actual decoding of the incoming VOPs. The tool is based on the trasmission of the statistic of the actual encoding algorithms, modes and parameters used to encode the incoming VOP. The 'cost' in complexity for the execution of each algorithm is measured or eastimated on each decoder platform. The actual statistic of the decoding algorithms is transmitted in the video bitstream and can be converted by means of the mentioned 'costs' into the VOP actual decoding cost for the specific decoder.

The tool is flexible since it enables, for each VO, the definition of the set of used statistics. Such definition is done in the VO header. The actual values of the defined statistics is then inserted into each VOP header according to the 'complexity estimation syntax'.

E.3 Resynchronization in Case of Unknown Video Header Format

Two video object layer starting indicators are supported:

1. video_object_layer_start_code, and
2. short_video_start_marker

The automatic detection of which of these byte aligned codes is present is unambiguous. The short_video_start_marker will never emulate a video_object_layer_start_code, since 23 byte-aligned zeros cannot occur in any video stream using the short_video_start_marker. The video_object_layer_start_code will also never emulate a short_video_start_marker, because its first non-zero bit is in a different location (provided byte alignment is not lost).

However, special attention needs to be paid if some application requires starting at any arbitrary point in a bitstream for which there is no prior knowledge of the format type. Although unlikely, a resync_marker can emulate a short_video_start_marker (for certain macroblock_number field lengths and macroblock_number values and vop_fcode_forward values).

Although the behavior of the decoder in these circumstances is not specified, it is suggested to perform validity checks on the first few bits beyond the short_video_start_marker if the video_object_layer_start_code is not the first starting indicator found. Numerous checks are possible, for example, checking the values of the bits 9, 10, 18-21 and 27 beyond the short_video_start_marker. The decoder may also choose to delay acquisition until an "I" vop-type is indicated in bit 17. Even simply discarding some data while searching for a video_object_layer_start_code prior to "timing out" with a decision to seek the short_video_start_marker may be acceptable for some applications.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex F
(informative)

Preprocessing and postprocessing

F.1 VOP Generation Tools: Automatic and Semi-automatic Segmentations

F.1.1 Automatic Segmentation

F.1.1.1 Introduction

The video coding scheme defined by this part of ISO/IEC 14496 offers several content-based functionalities, demanding the description of the scene in terms of so-called video-objects (VOs). The separate coding of the video objects may enrich the user interaction in several multimedia services due to flexible access to the bitstream and an easy manipulation of the video information. In this framework, the coder may perform a locally defined pre-processing, aimed at the automatic identification of the objects appearing in the sequence. Hence, segmentation is a key issue in efficiently applying the ISO/IEC 14496-2 coding scheme, although not affecting at all the bitstream syntax and thus not being a normative part of the standard.

Usually, the term segmentation denotes the operation aimed at partitioning an image or a video sequence into regions extracted according to a given criterion. In the case of video sequences, this partition should achieve the temporal coherence of the resulting sequence of object masks representing the video object. In the recent literature, different methods have been proposed for segmentation of video sequences, based on either a spatial homogeneity, a motion coherence criterion [13] or on joint processing of spatio-temporal [2][4][5][6][16]. These algorithms are expected to identify classes of moving objects, according to some luminance homogeneity and motion coherence criterion.

In this annex, a framework aiming at an appropriate combination of temporal and spatial segmentation strategies, developed throughout the standardisation phase of ISO/IEC 14496 Version 1, is described. The description is given only for informative purposes as the technique to extract objects from the scene is not standardised.

F.1.1.2 Description of a combined temporal and spatial segmentation framework

Throughout the work on automatic segmentation of moving objects, different proposals for temporal and spatial segmentation algorithms have been investigated. This resulted at the end in a combined temporal and spatial segmentation framework [11] which is shown in a high level block diagram in Figure F.1.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

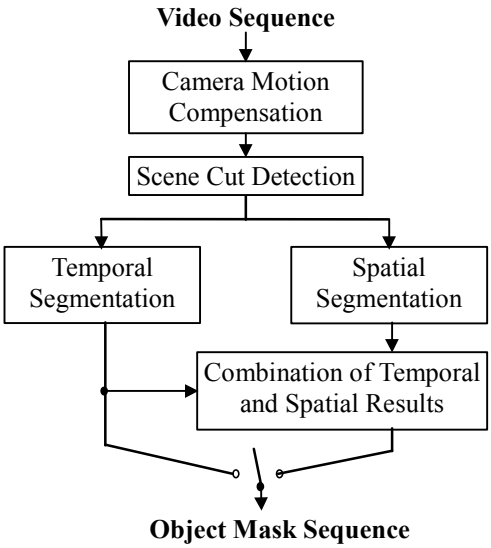


Figure F.1 — Block diagram of combined temporal and spatial segmentation framework

The combined scheme applies in a first step the general blocks of camera motion estimation and compensation [8][9][10] and scene cut detection [8][9][10] which can be seen as a kind of pre-processing in order to eliminate the influence of a moving camera. In a second step, either temporal or combined spatio-temporal segmentation of each image is carried out, depending on the requirements. The reason for this is, that in general only performing temporal segmentation requires less computational complexity. On the other hand, taking into account also spatial segmentation leads to more accurate segmentation results, but increases the computational complexity of the segmentation.

The evaluation of the segmentation results has been done subjectively by tape viewing for the MPEG-4 test sequences Mother Daughter, Akiyo, Hall Monitor, Container Ship, Coastguard, and Table Tennis, in QCIF and CIF formats with a frame rate of 10 Hz. In addition, the results have been objectively evaluated by the three criteria presented in [17], two for spatial accuracy and one for temporal coherence. The algorithms for camera motion estimation and compensation, for detection of scene-cuts, for temporal and spatial segmentation, and for combining temporal and spatial segmentation results will be described in the following.

F.1.1.2.1 Estimation and compensation of camera motion [8][9][10]

Given two successive images $I_{(k)}$ and $I_{(k+1)}$ of a video sequence captured by a static or moving camera, an apparent camera motion is estimated and compensated. For this purpose, a method is used which estimates the motion parameters of a rigid plane object within a 3D space. Thus, the background of the scene is assumed to be planar. The eight parameters of the motion model $a_1 - a_8$ can reflect any kind of motion, especially zoom and pan. For every pel (X_0, Y_0) in frame $I_{(k)}$ the corresponding pel (X_1, Y_1) in frame $I_{(k+1)}$ is given by:

$$X_1 = \frac{a_1 X_0 + a_2 Y_0 + a_3}{a_7 X_0 + a_8 Y_0 + 1} \qquad Y_1 = \frac{a_4 X_0 + a_5 Y_0 + a_6}{a_7 X_0 + a_8 Y_0 + 1}$$

The camera motion is estimated by regression considering only pels within background regions of the previous image. After the estimation of the motion parameters, the displacement vector for every pel is known. Then, a post-processing step counts for model failures in background regions due to the assumption of a rigid plane. By this post-processing step, the estimated displacement vector of every background pel is improved performing a full search within a squared area of limited size. For robustness reasons of the algorithm, the camera motion is only compensated if a moving camera has been detected.

F.1.1.2.2 Scene-cut detection [8][9][10]

In case of a scene cut between the two corresponding frames, the camera motion can not be compensated in a serious manner. Thus, scene cuts have to be detected for being able to perform further analysis steps on the video

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sequence. The scene cut detector described here evaluates whether the difference between the current original image $I_{(k+1)}$ and the camera motion compensated previous image exceeds a given threshold. The evaluation is only performed within the background regions of the previous frame.

F.1.1.2.3 Temporal segmentation based on change detection [10]

This temporal segmentation algorithm, which is mainly based on a change detection, can be subdivided into several steps, assuming that a possible camera motion has already been compensated: by the first step, an initial change detection mask (CDMi) between two successive frames is generated. In this mask, every pel where the luminance of the image has changed due to a moving object is marked. After that, boundaries of changed image areas are smoothed by a relaxation technique [1][2], resulting in a mask CDMs. Thereby the algorithm adapts frame-wise automatically to camera noise. While in [1] the variance of camera noise and the variance within object regions are both measured based on the CDMi, here the previous mask of object regions (OM) is additionally used, where every pel is marked which belongs to a moving object. In order to finally get temporally stable object regions, a memory for change detection masks (CDM) is applied, denoted MEM. The temporal depth of MEM adapts automatically to the sequence. Now, the CDMs is simplified by usage of a morphological closing-operator and elimination of small regions, resulting in the final CDM.

After that, an initial object mask (OMi) is calculated by eliminating the uncovered background areas from the CDM [7]. Therefore displacement information for pels within the changed regions is used. The displacement is calculated by a hierarchical block-matcher (HBM) [3]. For a higher accuracy of the calculated displacement vector field (DVF) the estimated CDM is considered. At last, the boundaries of the OMi are adapted to grey level edges in the corresponding image in order to improve the accuracy of the object mask, resulting in the final object mask (OM).

F.1.1.2.3.1 Computation of the initial CDM

Given two successive frames $I_{(k)}$ and $I_{(k+1)}$ of an image sequence, first the initial CDM (CDMi) is computed by a threshold operation on the squared luminance difference image, as it is described in [1]. The threshold is calculated by performing a significance test. Therefore the luminance difference image D is modelled as Gaussian camera noise with a variance σ^2 equal to twice the variance of the camera noise, which means that a luminance difference $\mathrm{d_i}$ for pel i is assumed to correspond to camera noise (null hypothesis H_0) and not to moving objects (hypothesis H_1):

$$p(d_i \mid H_0) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma^2}} \cdot \exp\left\{ -\frac{d_i^2}{2 \cdot \sigma^2} \right\} \tag{1}$$

Assuming that neighbouring pels are statistical independent, the sum of pels in the squared difference image within a window of size $(n \times n)$, normalised by twice the variance of camera noise, is χ^2 distributed with n^2 degrees of freedom (Eq. (2)).

$$T = \frac{1}{\sigma^2} \sum_{i=1}^{n\,n} d_i^2 \tag{2}$$

All pels for which the sum T exceeds a certain threshold $\mathrm{t_\alpha}$ are marked as changed, the others are marked as unchanged. The threshold $\mathrm{t_\alpha}$ is chosen in such a way, that assuming the pel has not changed caused by moving objects, the probability, that a calculated sum T is greater than this threshold, equals to a given value α.

$$\alpha = p(T > t_\alpha \mid H_0) \tag{3}$$

Here, the threshold t_α is equal to 220 in the case of QCIF sequences, while in the case of CIF sequences t_α is set to 165.

F.1.1.2.3.2 Relaxation of the initial CDM for spatial homogeneity

Because real objects have smooth contours, spatial homogeneity of the CDM is required. Spatial homogeneity is achieved by relaxation of the CDMi using a method [1]. There the CDMi is processed iteratively. In each iteration step a decision is made for every border pel in the CDMi, if it belongs to the changed or to the unchanged area.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Thus, for every border pel of changed image areas a local threshold is calculated taking into account the local neighbourhood of the pel. For the decision, if a pel has changed by moving objects or not a maximum a priori estimator (MAP estimator) is used:

$$\frac{P(Q_c \mid D)}{P(Q_u \mid D)} \underset{u}{\overset{c}{\gtrless}} 1 \Leftrightarrow \frac{p(D \mid Q_c)}{p(D \mid Q_u)} \underset{u}{\overset{c}{\gtrless}} \frac{P(Q_u)}{P(Q_c)} \tag{4}$$

Here, Q_c means that a pel is marked changed in the CDM, while Q_u means that this pel is marked unchanged in the CDM. *D* denotes the given difference image between frame $I_{(k)}$ and $I_{(k+1)}$. The probabilities in Eq. (4) are modelled as following:

The probability that a luminance difference d_i is caused by camera noise $p(D \mid Q_u)$ is described by the same Gaussian distribution as given in Eq. (1). The corresponding probability, that a luminance difference d_i is caused by moving objects $p(D \mid Q_c)$ is modelled as a Gaussian distribution, too, but the variance of this distribution is not twice the variance of the camera noise, but the variance σ_c^2 of the difference image D measured in foreground regions.

$$p(D \mid Q_u) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{d_i^2}{2\sigma^2}\right\} \quad (5), \qquad p(D \mid Q_c) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{d_i^2}{2\sigma_c^2}\right\} \tag{6}$$

The two a priori probabilities in Eq. (4) are modelled by a Markov random field considering a *(3x3)* -neighbourhood of a pel as shown in Figure F.2:

$$P(Q_u) = \frac{1}{Z} \exp\{-E(Q_u)\} \quad (7), \qquad P(Q_c) = \frac{1}{Z} \exp\{-E(Q_c)\} \tag{8}$$

The constant *Z* is for normalisation, while *E* denotes an energy term defined as:

$$E(Q_{u/c}) = v_B(u/c)B + v_C(u/c)C \tag{9}$$

The terms $v_B(q_k)$ and $v_C(q_k)$, $q_k \in \{u,c\}$, are a measure for the inhomogeneity of the neighbourhood of pel *k*, which is separated into horizontal or vertical neighbours (potential *B*) and diagonal neighbours (potential *C*), as shown in Figure F.2. So, the term $v_B(q_k)$ denotes the number of horizontal and vertical neighbours of pel *k* with the opposite label to q_k. In the same way the term $v_C(q_k)$ denotes the number of diagonal neighbours of pel *k* with the opposite label to q_k.

C	**B**	**C**
B	**X**	**B**
C	**B**	**C**

Figure F.2 — Neighbourhood of pel *X* and potentials *B* and *C* of the neighbours

From Eq. (4) to Eq. (9) the following decision rule can be derived, where d_i^2 denotes the squared luminance difference of pel *i*:

$$d_i^2 \underset{u}{\overset{c}{\gtrless}} 2 \cdot \frac{\sigma_c^2 \cdot \sigma^2}{\sigma_c^2 - \sigma^2} \cdot \left(\ln \frac{\sigma_c}{\sigma} + \left(v_B(c) - v_B(u)\right) \cdot B + \left(v_C(c) - v_C(u)\right) \cdot C \right) \tag{10}$$

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The label of every border pel in the CDMi is decided by using this decision rule. The rule should read as follows: if d_i^2 exceeds the threshold term on the right hand side of Eq. (10), the pel is set to changed $(\mathrm{CDMs}_{(k+1)}(i) := 1)$, otherwise it is set to unchanged $(\mathrm{CDMs}_{(k+1)}(\mathrm{i}) := 0)$. The relaxation is processed iteratively, until nothing has changed by relaxation or the maximal number of iteration steps *N* is reached. The so obtained CDM is denoted CDMs.

For calculating the CDM the variances of the static background σ^2 and within object regions ${\sigma_c}^2$ are needed. While in [1] always the CDMi is used for calculating the variances ${\sigma_c}^2$ and σ^2, it is more exact to calculate the variance σ^2 within the unchanged regions of the CDM and the variance ${\sigma_c}^2$ within the object regions of the OM. Thus, the variance ${\sigma_c}^2$ does not include uncovered background. Since the positions of static background, uncovered background and moving objects are still unknown for the current frame, in this approach the OM and CDM from the previous frame are used. In order to increase the robustness in case of noise the measured variances are averaged with those of the last three times.

In the energy term (Eq. (9)) the potential B is set to 5 and the potential *C* is set to 2.5. Further, the maximal number of relaxation steps *N* is set to 20.

F.1.1.2.3.3 Temporal homogeneity of the OM

In order to finally get temporal stable object regions, the CDMs is updated with the previous OM, i.e. in the CDMs additionally all pels are set to changed which belong to the OM of the previous frame. This is based on the assumption that all pels which belonged to the previous OM should belong to the current change detection mask. However, in order to avoid infinite error propagation, a pel from the previous OM is only labelled as changed in the CDMs, if it was also labelled as changed in the CDMs of one of the last L frames, too. For this purpose, a storage for every pel is applied, building a memory (MEM). The value of this storage indicates, if the respective pel was set to changed in one of the *L* previous CDMs; the value L denotes the depth of the memory. Considering that CDMs, OM and MEM are two-dimensional fields and that MEM is the zero-matrix for the first frame, the update of MEM can be formulated as:

$$MEM_{(k+1)}(x,y) = \begin{cases} L & ,if\ \mathrm{CDMs}_{(k+1)}(x,y) = 1 \\ \max(0, MEM_{(k)}(x,y) - 1) & ,if\ \mathrm{CDMs}_{(k+1)}(x,y) = 0 \end{cases} \tag{11}$$

The current CDMs is then updated by logical OR operation between CDMs and the previous OM, taking into account the memory MEM:

$$\mathrm{CDM}u_{(k+1)}(x,y) = C\mathrm{DMs}_{(k+1)}(x,y) \vee \begin{cases} \mathrm{OM}_{(k)}(x,y) & ,if\ MEM_{(k+1)}(x,y) > 0 \\ 0 & ,if\ MEM_{(k+1)}(x,y) = 0 \end{cases} \tag{12}$$

It is reasonable to set the depth of the memory L to a large value, if the motion of the moving objects in the scene is small, and to set L to a small value, if the motion of the objects is large. Thus, L shall be decreased with the average amplitude D of the displacement vectors measured within object regions. Further, it is reasonable to increase the depth of the memory L with the size S of the moving objects. Hence, it is necessary for automatic segmentation to adapt the value of L with the sequence. Therefore, the value of L is heuristically calculated:

$$L = \max\left(1, L_{\max} - (D^4 / c_{L_1}) - c_{L_2} \cdot \exp\left(-10^{-3} \cdot (S / c_{L_3})\right)\right) \tag{13}$$

As can be seen in Eq. (13) the values of *L* are within the interval $[1; L_{\max}]$. L is decreased by the two terms D^4 / c_{L_1} and $c_{L_2} \exp\left(-10^{-3}\, s / c_{L_3}\right)$. The first term increases with the average amplitude of the displacement vectors within object regions powered by 4. The second term decreases exponentially with the size of the object regions. While the constants $\mathrm{c}_{\mathrm{L}_1}$ and $\mathrm{c}_{\mathrm{L}_3}$ are for normalisation to different image formats, $\mathrm{c}_{\mathrm{L}_2}$ is a weighting factor.

The maximum depth of MEM $L_{\max}$ is set to 34. While in case of QCIF sequences $\mathrm{c}_{\mathrm{L}_1} = 1$ and $\mathrm{c}_{\mathrm{L}_3} = 1$, in case of CIF sequences $\mathrm{c}_{\mathrm{L}_1} = 15$ and $\mathrm{c}_{\mathrm{L}_3} = 4$. The weighting factor $\mathrm{c}_{\mathrm{L}_2}$ is set to 80.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

F.1.1.2.3.4 Simplification of the CDM

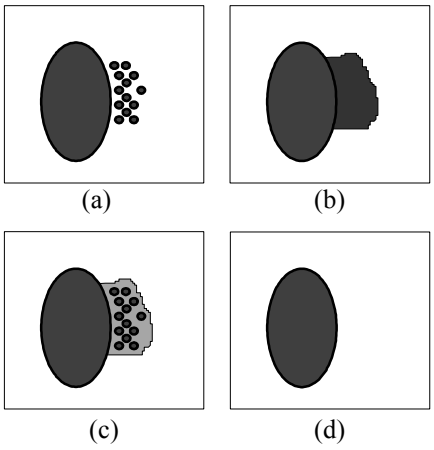


**Figure F.3 — Example for elimination of small regions using a ternary mask:
(a) CMD with a large region neighboured by smaller regions caused by noise;
(b) CMD after elimination of small regions without ternary mask;
(c) Ternary mask;
(d) CMD after elimination of small regions using a ternary mask**

In order to simplify the regions in the CDM, the morphological closing-operator is performed. Now, for elimination of small regions a ternary mask is generated, in which pels are labelled as changed, if they are set in the CDM before and after the closing operation. Pels, which are not set in the CDM before and after the closing operation, are set to unchanged in the ternary mask, while those pels, which are of different labels before and after the closing operation, are set to a third label. Within this ternary mask small changed and unchanged regions with a size below a certain threshold are eliminated. In order to prevent that very small regions, which are mostly due to noise spots, grow together with larger regions when the closing is performed, the third label is treated in a special way. An example for this method is given in Figure F.3a where a CDM is shown containing a large region, which is neighboured by many small regions caused by noise. Figure F.3b presents the CDM after closing is performed. It can be seen, that the noisy areas are grown together with the large region. In order to avoid this, the ternary mask is generated as described above and as it is shown for the example in Figure F.3c. The CDM after elimination of small regions by using a ternary mask is presented in Figure F.3d. At this time the final CDM is processed.

F.1.1.2.3.5 Detection and elimination of uncovered background

Based on the generated CDM the OMi is calculated by detecting and eliminating uncovered background, taking into account the displacement for pels within the calculated CDM. Therefore a DFV for the changed image areas is generated by hierarchical block-matching [3]. Hierarchical block-matching is used since the real motion is needed. In order to improve the DVF at object boundaries, the matching criterion is only evaluated for changed regions.

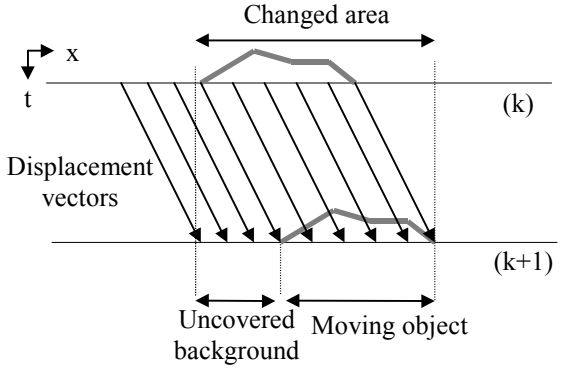


Figure F.4 — Separation of changed areas into foreground and uncovered background

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Now, pels are set to foreground, if foot- and top-point of the corresponding displacement vector are both inside the changed area in the current CDM [7]. If not, these pels are set to background. An example for that is given in Figure F.4, where an object moves from the left to the right. Further, to prevent moving shadows being misclassified as moving objects or parts of moving objects, moving shadows can be explicitly detected.

F.1.1.2.3.6 Adaptation of the OM to grey level edges

At last, the OMi is adapted to grey level edges in the current frame, resulting in the final OM. For that an array for correction is calculated: for every region in the OMi the outside limit of the area for correction is given by the contour of that region considering the OMi after the morphological blow operator has been performed two times. The inside limit of the area for correction is given by considering a width of correction of 6 pel. Within this area of correction the luminance edges of the current image are calculated using a Sobel operator.

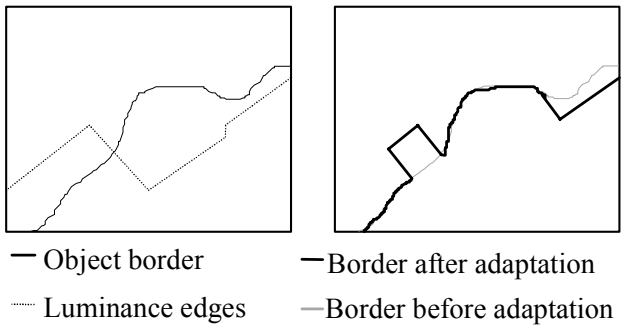


Figure F.5 — Exemplary result of the edge adaptation algorithm

By this edge adaptation technique every border pel in the OM is warped to the nearest grey level edge of frame $I_{(k+1)}$, if there exists such an edge in the area for correction within a circle of radius R = 12 around that pel.

In Figure F.5, an example for edge adaptation is given: On the left side a part of the border of an moving object in an OM is shown before edge adaptation is processed. Additionally the found luminance edges of the current image are drawn in. On the right side the corresponding part of the border after edge adaptation is presented.

F.1.1.2.4 Spatial segmentation based on watershed algorithm [5]

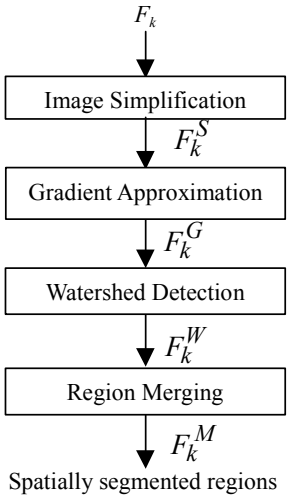


Figure F.6 — Spatial segmentation process

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Spatial segmentation splits the entire image into homogeneous regions in terms of intensity. The different homogeneous regions are distinguished by their encompassing boundaries that can be obtained from the spatial segmentation. A method for spatial segmentation is here introduced which is based on a morphological segmentation that utilizes morphological filters and a watershed algorithm [15]. However the watershed usually introduces over-segmentation problems which cause many partitioned regions even in a single object region. A region merging process is devised to avoid the over segmentation after watershed in the spatial segmentation. Figure F.6 depicts the spatial segmentation process. The segmentation algorithm consists of the following four steps: Simplification, Gradient approximation, Watershed detection and Region merging. The details of each step are explained in the following subclauses.

F.1.1.2.4.1 Image Simplification

For spatial segmentation, images are first simplified for the ease of spatial segmentation. Many morphological filters are often designed for smoothing noisy gray level images by a determined composition of opening and closing with a given structuring element [14]. Here erosion and dilation with a flat structuring element are used in constructing a morphological filter for image simplification.

$$\varepsilon_n(F_k(x,y)) = \min(F_k(x+l, y+m)), \quad \forall(l,m) \in R_n \tag{14}$$

$$\delta_n(F_k(x,y)) = \max(F_k(x-l, y-m)), \quad \forall(l,m) \in R_n \tag{15}$$

Morphological opening and closing operations are given by

$$\gamma_n(F_k) = \delta_n(\varepsilon_n(F_k)) \tag{16}$$

$$\varphi_n(F_k) = \varepsilon_n(\delta_n(F_k)) \tag{17}$$

It can be noticed that the morphological opening operation $\gamma_n(F_k)$ simplifies the image by removing the bright components that do not fit within the structuring elements while $\varphi_n(F_k)$ eliminates the dark components compared to the structuring components. For the case of the images containing fluctuating scale values, the bright and dark components should be considered for the simplification. With the boundary preservation, morphological opening/closing by reconstruction filters are used for the purpose of image simplification [14]. That is, the usage of these filter pairs remove regions that are smaller than a given size but preserve the contours of the remaining objects in the image. The opening/closing by reconstruction filters is given by partial reconstruction:

morphological opening by reconstruction of opening: $\gamma^{(rec)}(\gamma_n(F_k), F_k)$ (18)

morphological closing by reconstruction of closing: $\varphi^{(rec)}(\varphi_n(F_k), F_k)$. (19)

From (18) and (19), the image F_k is simplified by $\gamma_n(F_k)$ which operation results in the maximum value among the inspection region compared to the structural element values while $\varphi_n(F_k)$ results in the minimum value at the locations (x, y).

By removing unwanted details in texture of the regions, the image is homogenized in terms of region textures otherwise small connections among the different parts of the image lead to undesirable structures. So this morphological operation helps spatial segmentation obtain more semantic region partitioning in the later stages. The image simplification is not necessary applied only to the gray level image. For the regions that show low contrasts, the simplification may results in merging of the two regions that should be separated. The size of the regions eliminated is dependent on applications. A simplified image is shown in Figure F.8b.

F.1.1.2.4.2 Gradient Approximation [5]

The spatial gradient of the simplified image is approximated by the use of morphological gradient operator [14]. Through the image simplification, the inside of each homogeneous region is small gradient but large gradients are expected along the region boundaries among different homogeneous regions in the image. So the spatial gradient

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

can be used as an input of watershed algorithm to partition an image into homogeneous intensity regions. The morphological gradient operator used is expressed such as:

$$G(F_k) = \delta_1(F_k) - \varepsilon_1(F_k) \tag{20}$$

Note that the morphological gradient is positive. The positive gradients usually indicate borders between neighbor regions and allow the watershed to split the image into regions which are divided by the borders.

In the regions where intensity variations are small, adjacent regions are ambiguous so that watershed algorithm may yields false contours due to the resulting small gradients. To overcome this problem, we incorporate color information into gradient computation in which the largest values among the gradients obtained in αG_Y, βG_{Cr} and γG_{Cb} are chosen. α, β and γ are weighting factors applied to the gradients of αY, βC_r and γC_b, respectively.

$$\max\{\alpha G_Y, \beta G_{Cr}, \gamma G_{Cb}\} \tag{21}$$

In many practical cases, the resulting gradient images exhibits many noisy gradients which can cause the watershed algorithm to oversegment the image. In order to prevent this, the gradient images are thresholded by a value. That is, small gradient values less than the threshold value are set to zero, otherwise they remain same. This gradient image is used as input image for the watershed detection. A gradient image by the morphological gradient operator is shown in Figure F.8c.

F.1.1.2.4.3 Watershed detection [5][14][15]

The image is often interpreted as geographical surface in mathematical morphology and its grey level is treated as altitude. As for geographical surfaces, image structure exhibits inclines, hills and plateaus over the image plane. Usually a semantic region is distinguished by its inclines from the surrounding contiguous regions in the image. The gradient exhibits large values in the inclines. So a watershed algorithm is applied to the gradient image to partition the image into homogeneous intensity regions. A number of watershed algorithms have been proposed in the literatures but here we use an immersion based watershed algorithm that is simple and computationally efficient. The watershed algorithm is a region growing algorithm and it assigns pixels in the uncertainty area to the most similar region with some segmentation criterion such as difference of intensity values. The final result of the algorithm is a tessellation of the input image into its different catchment basins.

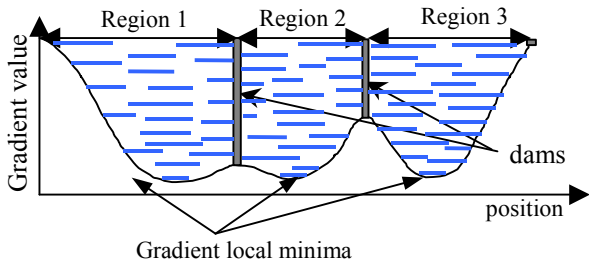


Figure F.7 — Watershed flooding

Figure F.7 depicts a 1-D graphical illustration of the watershed algorithm in an intuitive way. The local gradient minima are detected and used as seeds for region growing. In the image, these local minima are derived from homogeneous intensity regions. As the tessellation of the gradient image is immersed, each catchment basin is filled up from its lowest altitude which was selected as a seed. When the surface of the water in each catchment basin reach top of the crest, a dam is made for the water to overflow into the nearby catchment basins. Finally the water surface in each catchment basin reaches the same level but is confined within the region which is encompassed by the crest lines or dams built on the crest in the gradient image. Then each region is assigned a unique label.

The watershed algorithm is highly sensitive to gradient noise, which yields many catchment basins, so resulting in over-segmentation. To solve this problem, the region merging algorithm based on intensity homogeneity is incorporated. The image after the watershed detection is shown in Figure F.8d.

© ISO/IEC 2004 – All rights reserved

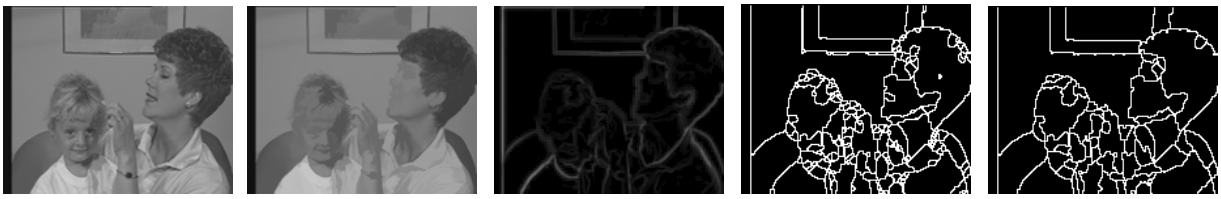
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

F.1.1.2.4.4 Region merging [5]

In the region merging step of the algorithm, small regions are merged in order to yield larger and meaningful regions which are homogeneous and different from its neighbors. This scheme represents both regions and their boundaries explicitly, and makes the storing and indexing of their semantic properties possible. For region merging, it must be determined which regions are merged to their neighbors according to certain predefined criteria. In this experiment, we used as a similarity measure (*T*) the ASAD (Average Sum of Absolute Difference denoted by R_t) of a region under consideration between two successive frames and the average intensity (R_s) of the region in the current frame.

$$T = \beta R_s + (1-\beta)R_t \tag{22}$$

Each region that consists of less than *N* pixels should be merged to its most similar region. That is, we compare the similarity measure (*T*) of a region under consideration with its neighbors. Then the region under consideration is merged to a region where the difference between two similarity measures is smallest. For QCIF and CIF sequences we used β = 0.2 in our experiments. The value *N* controls the amount of meaningful regions in a segmented image. We start the merging process with *N* = 10, and repeat merging iteratively with increasing *N* by 10 until the value *N* reaches 50. Therefore the merging process is done four times. A merged image is shown in Figure F.8e.



a) original image b) simplified image c) gradient image d) watershed image e) merged image

Figure F.8 — Interim results in Spatial Segmentation process

F.1.1.2.5 Combination of temporal and spatial results (method 1) [5]

In the last step, the decision of foreground(FG)/background(BG) is made on the spatially segmented regions with object masks obtained form temporal segmentation in the image. A spatial region ratio and temporal region ratio are computed to make judgement on FG/BG decision.

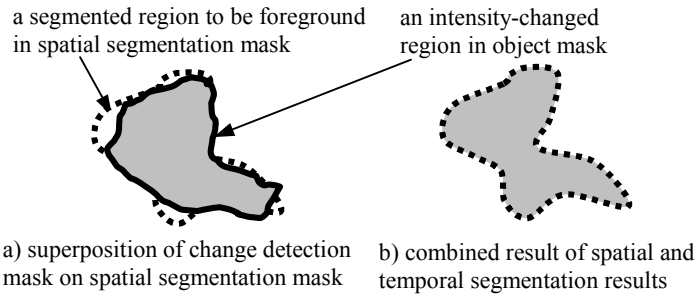


Figure F.9 — Combination of temporal and spatial segmentation results

First, we superimpose the change detection mask on top of the spatial segmentation mask. For the spatial region ratio, when a majority part of a segmented region belongs to changed region in the change detection mask, the whole area of the segmented region is declared as a foreground, otherwise a background. Figure F.9 illustrates a combination scheme of temporal and spatial segmentation results. The parts exceeding region boundaries in the spatially segmented region can be excluded from the foreground region in Figure F.9b when the portion of the exceeding pixels compared to the neighbour regions is small, while uncovered parts of the spatially segmented region by the object mask can be all foreground when their portion of areas are small compared to the part of the region covered by the foreground part of the object mask.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Now we consider a historical fact defined by the temporal region ratio. We overlay the spatially segmented regions on the previous labelled image. If the majority part of a region under consideration belongs to a foreground of the previous labelled image, the region is also considered a foreground. This step exploits historical attributes to correct the wrong results especially when a moving object stops.

F.1.1.2.6 Combination of temporal and spatial results (method 2)

This combing technique evaluates the results from the temporal and spatial segmentation. For time reasons, the described technique is considered as a preliminary version which will be further improved. The technique works as follows: for each region of the spatial segmentation a foreground / background decision is made. If at least 80% of the pels within a region were detected as foreground by the temporal segmentation, all pels of the region already detected as foreground and all pels within a correction range of 2 pel with respect to the boundary of the temporal segmentation are set to foreground, the other to background. If less than 80% of the pels within a region were detected as foreground by the temporal segmentation, all pels of the region already detected as background and all pels within a correction range of 2 pel with respect to the boundary of the temporal segmentation are set to background, all other to foreground. In future work, it is planned to additionally consider the size of the spatial regions and the quality of the temporal and spatial segmentation results in the combination criterion.

F.1.2 Semi-automatic Segmentation

In order to support the philosophy of the MPEG-4 Video standard, each image of video sequences should be represented in terms of VOPs to be encoded, that is, the objects to be encoded in image sequences should be prepared before the encoding process starts. The MPEG-4 Video has committed to providing an import tool to support for the object-based handling of video data during the phase 1 of MPEG-4 standardization.

However, the current technologies for automatic segmentation are somehow premature to obtain desirable segmentation results because automatic segmentation itself is an open-issue and ill-posed problem. Nevertheless, there is a more interesting and attractive approach to image segmentation which is semi-automatic. The approaches to semi-automatic segmentation are much more practical and give better segmentation results than the automatic segmentation.

In this informative annex, two different approaches are presented. The first method [18][19][20] consists of two processing steps: intra-frame segmentation and inter-frame segmentation. The intra-frame allows users to initially define any regions of interest via a graphical user interface and finds out precise boundaries of the region defined. The inter-frame segmentation automatically separates the region defined by object tracking techniques. The second method allows a user to define regions of interest by a rectangular box and to control user-supply parameters of the algorithm.

F.1.2.1 User-assisted segmentation of moving objects (method 1) [18][19][20]

As shown in Figure F.10, the semi-automatic segmentation can be thought of as a user-assisted segmentation technique. It consists of two steps: intra-frame segmentation and inter-frame segmentation. First, the intra-frame segmentation is applied in the first frame of the image sequence or the frames which contain newly appeared video objects or get bad segmentation results. The intra-frame segmentation can be processed by a user manually or semi-automatically. Then the inter-frame segmentation is applied in the next consecutive frames.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

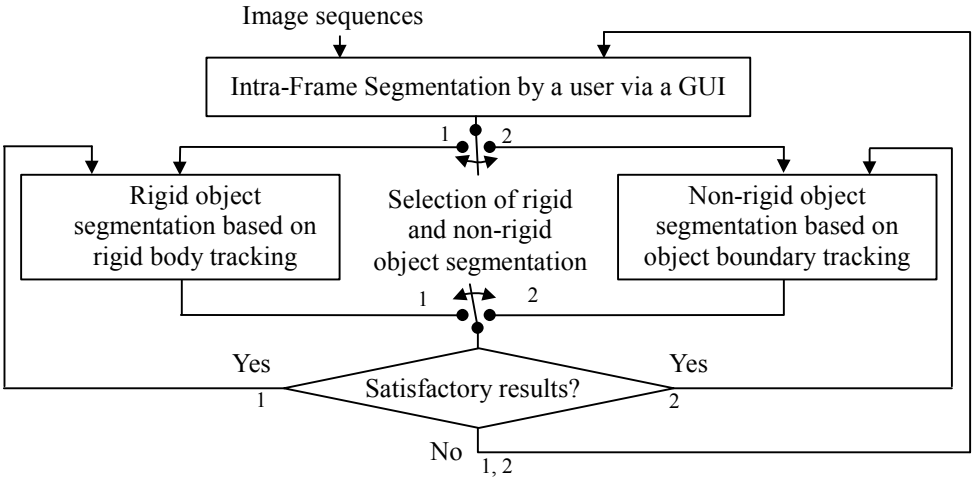


Figure F.10 — The block diagram of a user-assisted video object segmentation method

F.1.2.1.1 Intra-Frame Segmentation

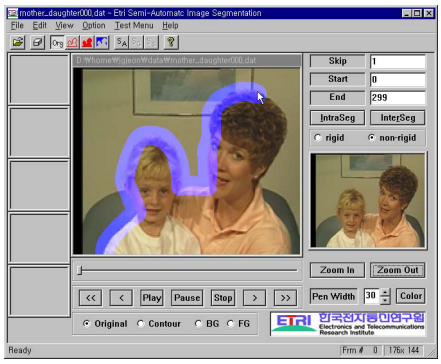


Figure F.11 — A graphical user interface for supporting intra-frame segmentation

The objective of intra-frame segmentation is to find precise object boundaries with cognitive information of Human at the beginning by user intervention. Since automatic segmentation separates moving objects as video objects without any cognitive information *a priori*, it is very difficult to obtain satisfactory segmentation results in general in generic video. Therefore much technological progress has to be made in order for automatic segmentation to be used in practice in real world problems. On one hand, since definition of a video object is somewhat subjective, user intervention is inevitable in segmentation process. That is, the human cognitive information can be provided into segmentation through a graphical user interface (GUI). Figure F.11 depicts an intra-frame segmentation.

Supposed the user is interested in extracting objects with precise boundaries of the objects in MOTHER_DAUGHTOR image sequences in the main window (a large window at the GUI center) of GUI in Figure F.11.

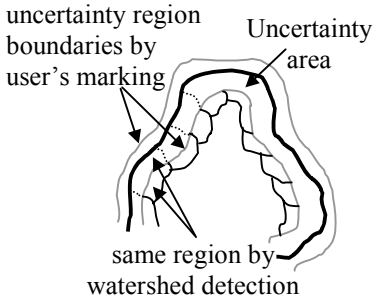


Figure F.12 — Findings of object boundaries with in the uncertainty area

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

A user can mark the entire objects all the way around the objects using a mouse with an 8-pixel wide width. A marked swath (light blue-colored region) made by the mouse constitutes an uncertainty area in which the real object boundaries are assumed to exists. A user can select a thickness of the mouse point in order to mark rough boundary regions around the true object boundaries of interest in an initial image. Figure F.13 illustrates the intra-frame segmentation process.

From the initial image with the marked region, globally labeled image is obtained in which the uncertainty region (UR) is labeled with '0', object regions of interest with '1' and the other regions with '2'. A watershed algorithm is then applied to the initial image in order to perform spatial segmentation.

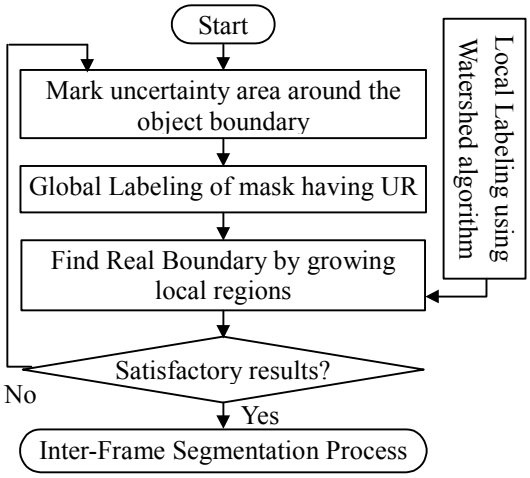


Figure F.13 — Intra-Frame Segmentation Process

The spatially segmented regions are labeled with integers, which forms a locally labeled image. In Figure F.13, locally labeled regions extending from the interior of the uncertainly region to the exterior are merged into a globally labeled region. That is, the regions that span between the uncertainty area and a globally labeled region are assigned into the globally labeled region in near vicinity. Finally the real boundaries of objects of interest can be obtained.

After finding the real boundary of the region of interest, the resulting resulting segmentation mask is taken as input into the inter-frame segmentation process if the intra-frame segmentaiton result is satisfactory. In the cases of unsatisfactory results for the real boundary finding, the intra-frame segmentation process can restart until the segmentation result is satisfactory. Note that any apriori initial segmentation mask can be used as input for the inter-frame segmentation process. This will be the case that the initial masks equivalent to the results of the intra-frame segmentaiton process can also be obtained from other intra-frame segmentation operations.

F.1.2.1.2 Inter-Frame Segmentation

In the inter-frame segmentation, the moving objects are segmented with precise object boundaries in automatic manners based on the object boundary tracking for non-rigid objects or the motion tracking for rigid objects. This automatic segmentation of the inter-frame segmentation alleviates user's effort of intervention during the segmentation process. Since the inter-frame segmentation starts with the initially obtained object mask having precise object boundary, it yields precise segmentation masks for the moving objects. In the following subclauses, the core parts of the inter-frame segmentation are described in details for non-rigid and rigid object segmentation.

F.1.2.1.2.1 Non-rigid object segmentation by object boundary tracking

The non-rigid object segmentation process is shown in Figure F.14. At first, the previous object boundary is extracted from the mask information obtained by intra-frame segmentation process. To obtain the initial object boundary in the current frame, the motion estimation and compensation is then applied to every coordinate composing the previous object boundary. Depending on the characteristics of the background, block or polygon matching criteria can be used to minimize the erroneous result due to noise.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

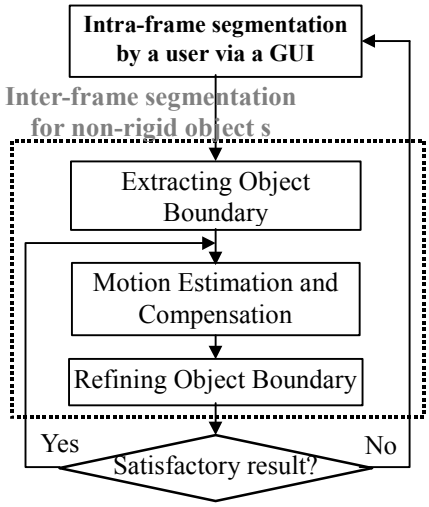


Figure F.14 — Inter-frame segmentation process for non-rigid object

$$MAD(u,v) = \sum_{i=-N/2}^{N/2} \sum_{j=-N/2}^{N/2} \left| S(i+u, j+v) - R(i,j) \right| \times \alpha(i,j)$$

$$MV = (u,v)\Big|_{\min MAD(u,v)}, \quad -p \le u,v \le p$$

where N is a reference block size and p is a search range. $S(i+u, j+v)$ and $R(i,j)$ are the search area data in the current frame and the reference block data in the previous frame, respectively. For the polygon based motion estimation $\alpha(i,j)$ is an alpha value of the corresponding pixel $R(i,j)$ while $\alpha(i,j)$ is always 1 for the block based motion estimation. Then the initial boundary is refined pixel by pixel by using both spatial and temporal information to reduce error propagation and the final object boundary is obtained. At this refinement step, the edge strength in the current frame is used as the spatial information and the motion compensated error is used as the temporal information.

$$MV_{refined} = (u+u', v+v')\Big|_{\max \frac{EdgeStrength(u+u', v+v')}{MAD(u+u', v+v')}}, \quad -p' \le u', v' \le p'$$

After refining the object boundary, a median filter is applied all the way around the object boundary in order to smooth out the rough boundary intervals.

F.1.2.1.2.2 Rigid object segmentation by motion tracking

In this subclause, the segmentation algorithm for rigid objects is described. It is based on parametric motion estimation and motion compensation which establish the mapping from one *rigid* object contour to the other. The segmentation algorithm consists of the following 4 steps: a) 2 parameter motion estimation; b) object mask refinement; c) 6 parameter motion estimation; d) motion compensation for the object contour. The block diagram is illustrated in Figure F.15.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

F.1.2.1.2.2.1 Object Motion Models

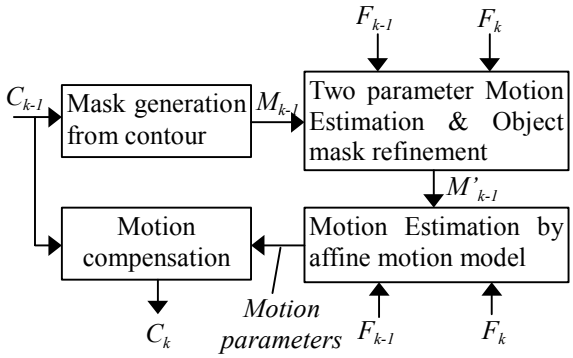


Figure F.15 — Block diagram for the object motion model

We use the affine 6-parameter motion model for the object motion estimation.

$$p(x, y, t) = a + bx + cy$$

$$q(x, y, t) = d + ex + fy$$

where $(p(x, y, t), q(x, y, t))$ is the displacement induced on pixel (x, y) by the motion of the object between frames t and $t+1$. Assuming the intensity is invariant, $I(x+p, y+q, t+1)$ is equal to $I(x, y, t)$. By use of Taylor expansion, following equation is obtained.

$$pI_x + qI_y + I_t = 0$$

where I_x, I_y, I_t are partial derivatives of the intensity $I(x, y, t)$ with respect to x, y, t.

$$E(p, q) = \sum_{(x,y)\in M} (pI_x + qI_y + I_t)^2$$

where M is the object mask. Deriving $E(p, q)$ with respect to the motion parameters and setting to zero yields six linear equations in the six unknowns: $\mathbf{m} = (a, b, c, d, e, f)^T$ as follows.

$$\begin{pmatrix} \sum I_x^2 & \sum xI_x^2 & \sum yI_x^2 & \sum I_xI_y & \sum xI_xI_y & \sum yI_xI_y \\ \sum xI_x^2 & \sum x^2I_x^2 & \sum xyI_x^2 & \sum xI_xI_y & \sum x^2I_xI_y & \sum xyI_xI_y \\ \sum yI_x^2 & \sum xyI_x^2 & \sum y^2I_x^2 & \sum yI_xI_y & \sum xyI_xI_y & \sum y^2I_xI_y \\ \sum I_xI_y & \sum xI_xI_y & \sum yI_xI_y & \sum I_y^2 & \sum xI_y^2 & \sum yI_y^2 \\ \sum xI_xI_y & \sum x^2I_xI_y & \sum xyI_xI_y & \sum xI_y^2 & \sum x^2I_y^2 & \sum xyI_y^2 \\ \sum yI_xI_y & \sum xyI_xI_y & \sum y^2I_xI_y & \sum yI_y^2 & \sum xyI_y^2 & \sum y^2I_y^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = -\begin{pmatrix} \sum I_xI_t \\ \sum xI_xI_t \\ \sum yI_xI_t \\ \sum I_yI_t \\ \sum xI_yI_t \\ \sum yI_yI_t \end{pmatrix}$$

Let the 8×8 matrix $\mathbf{A}$ and right side column vector $\mathbf{b}$.

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

F.1.2.1.2.2.2 Motion Estimation & Compensation

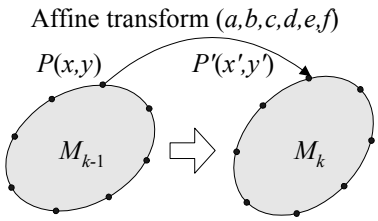


Figure F.16 — The object mask obtained by affine motion compensation

First, we generate the object mask M_{k-1} from previous contour C_{k-1}. Then we perform 2-parameter motion estimation between two successive frames F_{k-1}, F_k for the foreground object region. The object mask is refined by removing outlier points whose motion compensation error is larger than the given threshold. We calculate the affine motion parameters with the refined object mask. The motion parameter $\mathbf{m}$ is given by $\mathbf{A}^{-1}\mathbf{b}$. Since the displacement between two successive frames may not sufficiently small for the above equation, iterative estimation is needed to obtain correct motion parameter. Iterative estimation is as follows; 1) $\mathbf{m}^{(0)} \leftarrow \mathbf{0}^T$, $i \leftarrow 0$, 2) F_k is warped toward F_{k-1} using motion parameter $\mathbf{m}^{(i)}$, 3) calculate $\mathbf{A}, \mathbf{b}, \Delta\mathbf{m} = \mathbf{A}^{-1}\mathbf{b}$ between F_{k-1} and F_k^{warped}, 4) $\mathbf{m}^{(i+1)} = \mathbf{m}^{(i)} + \Delta\mathbf{m}$, $i \leftarrow i+1$, 5) if $\|\Delta\mathbf{m}\| < Th$ then stop iteration else go to step 2). All points in C_{k-1} are affine transformed and construct the new object contour C_k as shown in Figure F.16.

F.1.2.1.3 Results of the user-assisted segmentation

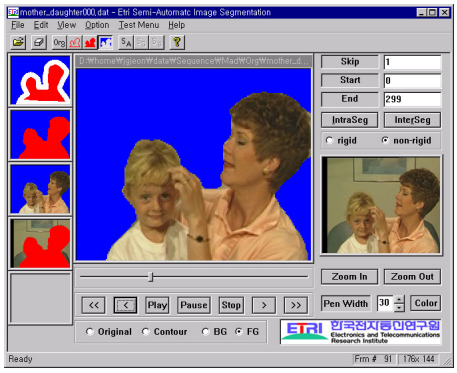


Figure F.17 — Results of the user-assisted segmentation

For the intra-frame segmentation, the top most window in the left column windows of the GUI displays a labelled image, called *mask image*, of the initially marked image by the user. The uncertainty region is remarked by the white swath and the globally labelled regions are indicated by red and blue colors. After finding the real object boundary, the resulting boundary image is shown in the second top most window. For the inter-frame segmentation, the non-rigid objects (foreground) are separated automatically and shown in the main window. These foreground and background images are also shown in the third and fourth windows in the left column windows. The foreground and background of the entire image can be used as video object planes for MPEG-4. Note that, in the user-assisted segmentation, the inter-frame segmentation process can be stopped at any frame by any metrics of measuring segmentation performance or user's judgement if the segmentation results are unsatisfactory.

F.1.2.2 Semi-automatic approach to segmentation (method 2)

Semi-automatic approach to segmentation could be a straight way to increase segmentation performance for non real time applications, enabling the user to introduce in the segmentation process high-level information.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2004 – All rights reserved
Not for Resale

A lot of interaction possibilities between the user and the segmentation process can be envisaged. For example, a user can initially mark objects of interest around the object boundaries; then the selected objects could be tracked during their temporal evolution in the video sequence until the next user interaction occurs. In this case, the user should be able to stop at any frame in image sequences and to revisit/remark the objects of interest with more exact boundaries whenever the segmentation performance is not satisfactory or a new object of interest has to be extracted.

Simple interactions could be useful to adapt segmentation procedures temporally (frame by frame), spatially (inside each frame) or sequence by sequence, by changing some significant parameter on the basis of the viewed results. The interactive adjustment of the parameters could be useful to discriminate more efficiently interesting objects from randomly varying background (see, grass, etc.).

A different approach enables the user to select, inside a frame, the area (Region Of Interest) in which the object is located. This allows to optimize the extraction process on the ROI only. Segmentation techniques based on motion analysis modules could take advantage from this approach as they usually detect several moving regions, but only one represents the object of interest for the user.

The algorithm for the semi-automatic segmentation separates, in each frame of an image sequence, regions corresponding to moving objects from regions belonging to a static background [12][21].

The algorithm is based on a change detection analysis performed along several frames in order to detect, for each frame, pixels that belong to moving objects. The motion detection is performed by tracking each pixel of the current frame, along a group of preceding frames in the space of the thresholded fourth order moment of the differences.

The algorithm [12][21], produces the segmentation map of each frame f_k of the sequence by processing a group of frames $\{f_{k-i},\ i=0,..n\}$. The number of frames n varies on the basis of the estimated object velocity [12]. Any global motion component is removed by a preprocessing stage, aligning f_{k-i} to f_k, $j=1,..n$. For each frame f_k, the algorithm splits in three steps (see Figure F.18).

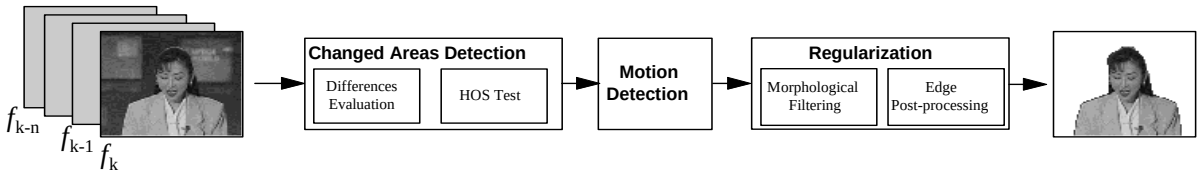


Figure F.18 — Segmentation Using Higher Order moments and motion tracking: Block Diagram

Step One: Changed areas detection. The differences $\{ d_{k-j}(x,y)=f_{k-j}(x,y)- f_{k-n}(x,y),\ j=0,..n-1 \}$ of each frame of the group with respect to the first frame f_{k-n} are evaluated in order to detect the changed areas, due to object motion, uncovered background and noise. In order to reject the luminance variations due to noise, an Higher Order Statistic test is performed. Namely, for each pixel (x,y) the fourth-order moment of each inter-frame difference $d(x,y)$ is estimated on a 3×3 window (see (23) and (24)), it is compared with a threshold adaptively set on the basis of the estimated background activity [12], and set to zero if it is below the threshold. On the sequence of the thresholded fourth-order moment maps $\tilde{m}^{(4)}_{d_{k-j}}(x,y)$, a motion detection procedure is performed.

$$\hat{m}_d(x,y)=\frac{1}{9}\sum_{(s,t)\in\eta(x,y)} d(s,t) \quad (23), \qquad \hat{m}^{(4)}_d(x,y)=\frac{1}{9}\sum_{(s,t)\in\eta(x,y)} (d(s,t)-\hat{m}_d)^4 \quad (24)$$

Step Two: Motion detection. This step aims at distinguish changed areas representing uncovered background (which stands still in the HOS maps) and moving objects (moving in the HOS maps). At the j-th iteration, the pair of thresholded HOS maps $\tilde{m}^{(4)}_{d_{k-j}}(x,y), \tilde{m}^{(4)}_{d_{k-j-1}}(x,y)$ is examined. For each pixel (x,y) the displacement is evaluated on a 3x3 window, adopting a SAD criterion (Sum of Absolute Differences), and if the displacement is not null the pixel is classified as moving. Then, the lag j is increased (i.e. the pair of maps slides) and the motion analysis is repeated, until $j=n-2$. Pixels presenting null displacements on all the observed pairs are classified as still. Note that, from a computational point of view, at each iteration, only pixels that were not already classified as moving are

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

examined. Moreover, the matching is not necessary on the pixels which are zero and are included in a 3x3 zero window, and they are assumed to be still. Thus, at the expense of some comparisons, the matching is performed on few pixels, and at the expense of examining more than a pair of difference, the search window is small.

Step Three: Regularization Still regions, internal to moving regions, are re-assigned to foreground. The regularization algorithm refines the imposing a priori local connectivity constraints, on both the background and the foreground.

Namely, topological constraints on the size of objects irregularities such as holes, isthmi, gulfs and isles are imposed by means of morphological filtering. Five morphological operations, with circular structuring element, are applied (Table F.1). From a topological point of view, the regularization supports multiple moving regions whose size exceeds the size of the structuring element, corresponding to different moving objects. Finally, a post-processing operation refines the results on the basis of spatial edges.

Introducing user interaction in the segmentation algorithm allows to enhance its capability to extract the shapes of moving objects with accuracy. In fact, to minimize false alarm errors, the objects of interest may be partially missed and in particular the extracted object borders usually do not match the real object borders. To increase the segmentation performance the following user-algorithm interactions are provided:

Table F.1 — Morphological operations

	Radius	Radius	Iteration	Iteration
Closing by Rec.	2	3	5	6
Opening by Rec.	7	12	20	35
Opening	4	6		
Closing	3	6		
Closing by Rec.	10	15	200	250

The user driven threshold setting in order to tune, sequence by sequence and frame by frame, the algorithm capability to select the objects of interest.

The user marked area in which the object of interest is located.

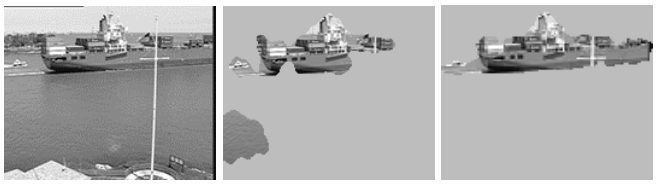


Figure F.19 — Segmentation results comparison: original, automatic, semiautomatic

In this case the application of the automatic segmentation technique can be restricted and optimized within that ROI only, and in most the cases this gives a performance enhancement because parameters critical for revealing the object are estimated on the region where the object is included and not on the entire image where regions with different characteristics are usually present.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

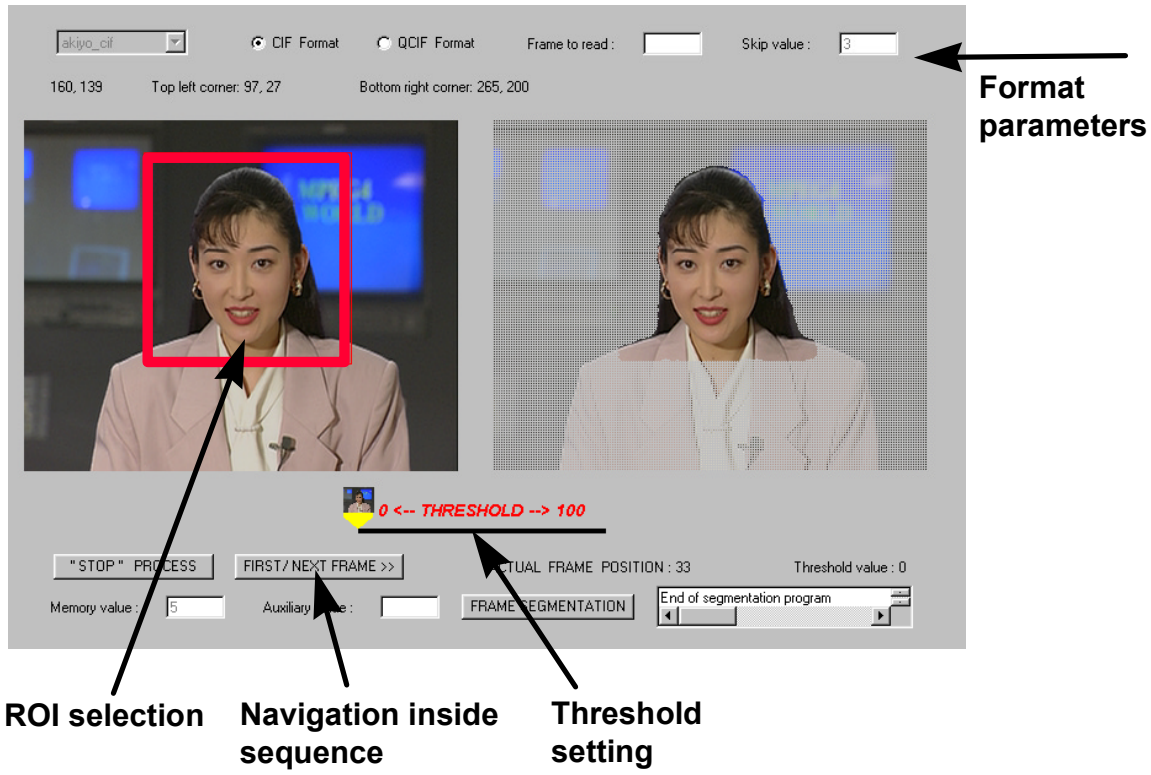


Figure F.20 — Java based Graphical User Interface for semi-automatic segmentation

An example of the obtained results is reported in Figure F.19, in comparison with the result obtained by a completely automatic technique. The enhanced algorithm capability to estimate critical parameters allows to improve its performance in revealing the object shape with accuracy.

A graphical user interface (GUI) allowing easy user interactions with the segmentation process is shown in Figure F.20. Basically, the result of the segmentation could be presented to the user frame by frame, so that he can decide to continue the process with the same parameters or to stop the process, revise the information he previously gave (such as parameters values or ROI selections), and restart the process.

F.1.3 References

[1] T. Aach, A. Kaup, R. Mester, "Statistical model-based change detection in moving video," *Signal Processing*, Vol. 31, No. 2, pp. 165-180, March 1993.

[2] T. Aach, A. Kaup, R. Mester, "Change detection in image sequences using Gibbs random fields: a Bayesian approach," *Proceedings Int. Workshop on Intelligent Signal Processing and Communication Systems*, Sendai, Japan, pp. 56-61, October 1993.

[3] J. Benois, L. Wu, "Joint Contour-based and Motion-based Image Sequences Segmentation for TV Image Coding at Low Bit-rate," *Visual Communications and Image Processing*, Chicago, Illinois, September 1994.

[4] M. Bierling, "Displacement estimation by hierarchical blockmatching," *3rd SPIE Symposium on Visual Communications and Image Processing*, Cambridge, USA, pp. 942-951, November 1988.

[5] P. Bouthemy, E. François, "Motion segmentation and qualitative dynamic scene analysis from an image sequence," *International Journal of Computer Vision,* vol.10, no.2, pp157-182, 1993.

[6] J. Dugelay, H. Sanson, "Differential methods for the identification of 2D and 3D motion models in image sequences," *Signal Processing*, vol.7, pp. 105-127, Sept. 1995.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[7] C. Gu, T. Ebrahimi, M. Kunt, "Morphological Spatio-temporal Segmentation for Content-based Video Coding," *International Workshop on Coding Techniques for Very Low Bit-rate Video*, Tokyo, Japan, November 1995.

[8] M. Hötter, R. Thoma, "Image Segmentation based on object oriented mapping parameter estimation," *Signal Processing*, vol. 15, No. 3, pp. 315-334, October 1988.

[9] R. Mech, M. Wollborn, "A Noise Robust Method for Segmentation of Moving Objects in Video Sequences," *International Conference on Acoustic, Speech and Signal*, Munich, Germany, April 1997.

[10] R. Mech, M. Wollborn, "A Noise Robust Method for 2D Shape Estimation of Moving Objects in Video Sequences Considering a Moving Camera," *Workshop on Image Analysis for Multimedia Interactive Services*, Louvain-la-Neuve, Belgium, June 1997.

[11] R. Mech, M. Wollborn, "A Noise Robust Method for 2D Shape Estimation of Moving Objects in Video Sequences Considering a Moving Camera," *Signal Processing: Special Issue on Video Sequence Segmentation for Content-based Processing and Manipulation*, Vol. 66, No. 2, pp. 203-217, April 1998.

[12] A. Neri, S. Colonnese, G. Russo, "Video Sequence Segmentation for Object-based Coders using Higher Order Statistics," *ISCAS* '97, Hongkong, June 1997.

[13] A. Neri, S. Colonnese, G. Russo, "Automatic Moving Objects and Background Segmentation by means of Higher Order Statistics," *IS&T Electronic Imaging* '97 Conference: Visual Communication and Image Processing, San Jose, 8-14 February 1997, SPIE Vol. 3024.

[14] F. Pedersini, A. Sarti, S. Tubaro, "Combined Motion and Edge Analysis for a Layer-based Representation of Image Sequences," *IEEE International Conference on Image Processing*, Lausanne, Switzerland, September 1996.

[15] P. Salembier, M. Pardàs, "Hierarchical Morphological Segmentation for Image Sequence Coding," *IEEE Transactions on Image Processing*, Vol.3, No.5, pp. 639-651, September 1994.

[16] Luc Vincent, Pierre Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on PAMI*, Vol.13, No. 6, pp. 583-598, June 1991.

[17] N. T. Watsuji, H. Katata, T. Aono, "Morphological Segmentation with Motion Based Feature Extraction," *International Workshop on Coding Techniques for Very Low Bit-rate Video*, Tokyo, Japan, November 1995.

[18] *Core Experiments on Multifunctional and Advanced Layered Coding Aspects of MPEG-4 Video*, ISO/IEC JTC 1/SC 29/WG 11 N 2176, May 1998.

[19] Jae Gark Choi, Munchurl Kim, Myoung Ho Lee and Chieteuk Ahn, "A user-assisted segmentation method for video object plane generation," *1998 International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC'98)*, Sokcho, Korea, pp. 7 – 10, July 1998.

[20] Jinsuk Kwak, Jun Geun Jeon, Munchurl Kim, Myoung Ho Lee and Chieteuk Ahn, "A semi-automatic video segmentation method for object oriented multimedia applications," *1999 International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC'99)*, Niigata, Japan, pp. 114 – 117, July 13-15, 1999.

[21] Juergen Stauder, Roland Mech, and Joern Ostermann, "Detection of moving Cast Shadows for Object Segmentation," to be published in *IEEE Transactions on Multimedia*, vol. 1, no. 1, 1999.

F.2 Bounding Rectangle of VOP Formation

This clause describes the bounding rectangle of VOP formation process. The formation of the bounding rectangle of VOP is based on the segmented shape information. The following explains the process to achieve the bounding rectangle of a VOP in such a way that the minimum number of macroblocks containing the object will be attained in order to achieve a higher coding efficiency.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1. Generate the tightest rectangle whose vertical and horizontal positions of the top-left point are even numbered. In case of interlaced mode, the vertical position of the top-left point of the rectangle is a multiple of 4.

2. If the top left position of this rectangle is not the same as the origin of the image, the following steps have to be performed. Otherwise no further processing is necessary.

3. Form a control macroblock at the top left corner of the tightest rectangle as shown in Figure F.21.

4. Count the number of macroblocks that completely contain the VOP for all the points of which vertical and horizonal positions are even numbered (in case of interlaced mode, all the points of which horizontal positions are even numbered and vertical positions are multiple of 4) of the control macroblock using the following procedure:

   - Generate a bounding rectangle from the control point to the right bottom side of the VOP which consists of multiples of 16x16 blocks.

   - Count the number of macroblocks in this bounding rectangle, which contain at least one object pel. To do so, it would suffice to take into account the boundary pels of a macroblock only.

5. Select the control point that results in the smallest number of non transparent macroblocks for the given object.

6. Extend the top left coordinate of the tightest rectangle generated in Figure .F.21 to the selected control coordinate. This will create a rectangle that completely contains the object but with the minimum number of non transparent macroblocks in it. The VOP horizontal and vertical spatial references are taken directly from the modified top-left coordinate.

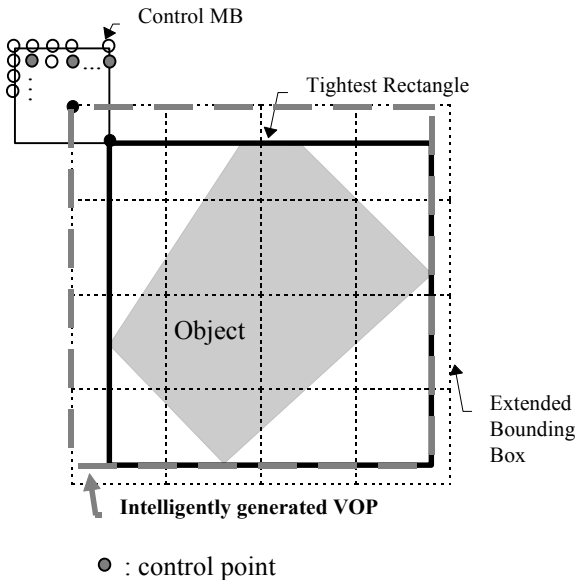


Figure F.21 — Intelligent VOP Formation

F.3 Postprocessing for Coding Noise Reduction

The post-filter consists of deblocking filter and deringing filter. Either one or both of them can be turned on as needed.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

F.3.1 Deblocking filter

The filter operations are performed along the 8x8 block edges at the decoder as a post-processing operation. Luminance as well as chrominace data is filtered. Figure F.22 shows the block boundaries.

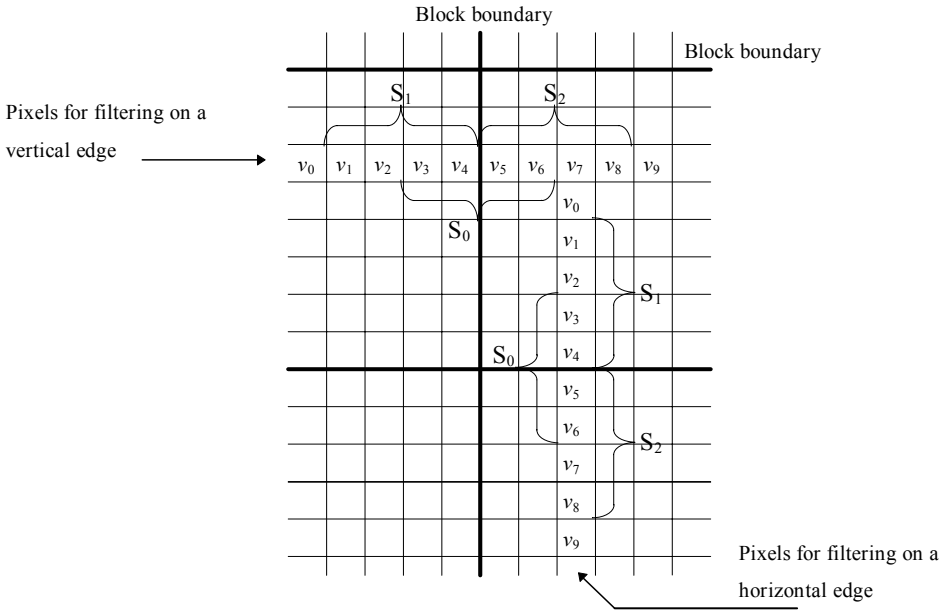


Figure F.22 — Boundary area around block of interest

In the filter operations, two modes are used separately depending on the pixel conditions around a boundary. The following procedure is used to find a very smooth region with blocking artifacts due to small DC offset and to assign it a DC offset mode. In the other case, default mode operations are applied.

eq_cnt = ϕ(v0–v1) + ϕ(v1–v2) + ϕ(v2–v3) + ϕ(v3–v4) + ϕ(v4–v5) + ϕ(v5–v6) + ϕ(v6–v7)
+ ϕ(v7–v8) + ϕ(v8–v9),
where $\phi(\gamma) = 1$ if $|\gamma| \leq$ THR1 and 0 otherwise.
If (eq_cnt $\geq$ THR2)
DC offset mode is applied,
else
Default mode is applied.

For the simulation, threshold values of THR1 = 2 and THR2 = 6 are used.

In the default mode, a signal adaptive smoothing scheme is applied by differentiating image details at the block discontinuities using the frequency information of neighbor pixel arrays, S_0, S_1, and S_2,. The filtering scheme in default mode is executed by replacing the boundary pixel values v_4 and v_5 with v_4' and v_5' as follows:

$$v_4' = v_4 - d,$$

$$v_5' = v_5 + d,$$

and $$d = \text{CLIP}(5\cdot(a_{3,0}' - a_{3,0})//8,\ 0,\ (v_4 - v_5)/2) \cdot \delta(|a_{3,0}| < QP)$$

where $$a_{3,0}' = \text{SIGN}(a_{3,0}) \cdot \text{MIN}(|a_{3,0}|, |a_{3,1}|, |a_{3,2}|).$$

Frequency components $a_{3,0}$, $a_{3,1}$, and $a_{3,2}$ can be evaluated from the simple inner product of the approximated DCT kernel [2 -5 5 -2] with the pixel vectors, i.e.,

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

a3,0 = ([2 -5 5 -2] • [v3 v4 v5 v6]T) // 8,
a3,1 = ([2 -5 5 -2] • [v1 v2 v3 v4]T) // 8,
a3,2 = ([2 -5 5 -2] • [v5 v6 v7 v8]T) // 8.

Here CLIP(*x,p,q*) clips *x* to a value between *p* and *q*; and QP denotes the quantiser scale of the macroblock where pixel v_5 belongs. δ(*condition*)=1 if the "*condition*" is true and 0 otherwise.

In very smooth region, the filtering in the default mode is not good enough to reduce the blocking artifact due to DC offset. So we treat this case in the DC offset mode and apply a stronger smoothing filter as follows :

max = MAX (v1, v2, v3, v4, v5, v6, v7, v8),
min = MIN (v1, v2, v3, v4, v5, v6, v7, v8),
if (|max–min| < 2·QP) {

$$v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \le n \le 8$$

$$p_m = \begin{cases} \left(\left|v_1 - v_0\right| < QP\right) ? v_0 : v_1, if \quad m < 1 \\ v_m, \qquad\qquad if 1 \le m \le 8 \\ \left(\left|v_8 - v_9\right| < QP\right) ? v_9 : v_8, if \quad m > 8 \end{cases}$$

$$\{b_k : -4 \le k \le 4\} = \{1,1,2,2,4,2,2,1,1\} // 16$$

}
else
No change will be done.

The above filter operations are applied for all the block boundaries first along the horizontal edges followed by the vertical edges. If a pixel value is changed by the previous filtering operation, the updated pixel value is used for the next filtering.

F.3.2 Deringing filter

This filter comprises three subprocesses; threshold determination, index acquisition and adaptive smoothing. This filter is applied to the pixels on 8x8 block basis. More specifically 8x8 pixels are processed by referencing 10x10 pixels at each block. The following notation is used to specify the six blocks in a macroblock. For instance, block[5] corresponds to the *Cb* block whereas block[*k*] is used as a general representation in the following subclauses.

F.3.2.1 Threshold determination

Firstly, calculate maximum and minimum gray value within a block in the decoded image. Secondary, the threshold denoted by *thr[k]* and the dynamic range of gray scale denoted by *range[k]* are set:

$$thr[k] = \left(\max imum[k] + \min imum[k] + 1\right) / 2$$

$$range[k] = \max imum[k] - \min imum[k]$$

An additional process is done only for the luminance blocks. Let *max_range* be the maximum value of the dynamic range among four luminance blocks.

$$\max_ range = range[k_{\max}]$$

Then apply the rearrangement as follows.

```
for( k=1 ; k<5 ; k++ ){
    if( range[k] < 32 && max_range > =64 )
        thr[k] = thr[kmax];
    if( max_range<16 )
        thr[k] = 0;
}
```

--`,,-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

F.3.2.2 Index acquisition

Once the threshold value is determined, the remaining operations are purely 8x8 block basis. Let *rec(h,v)* and *bin(h,v)* be the gray value at coordinates *(h,v)* where *h,v=0,1,2,...,7*, and the corresponding binary index, respectively. Then *bin(h,v)* can be obtained by:

$$bin(h,v) = \begin{cases} 1 & if\ rec(h,v) \geq thr \\ 0 & otherwise \end{cases}$$

Note that *(h,v)* is use to address a pixel in a block, while *(i,j)* is for accessing a pixel in a 3x3 window.

F.3.2.3 Adaptive smoothing

F.3.2.3.1 Adaptive filtering

The figure below is the binary indices in 8x8 block level, whereas practically 10x10 binary indices are calculated to process one 8x8 block.

	0	0	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	0	0	
	1	0	0	0	0	0	0	0	1	1	
	1	1	0	0	0	0	0	1	1	1	
	1	1	1	0	0	0	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	0	
	1	1	1	1	1	1	1	1	0	0	

Figure F.23 — Example of adaptive filtering and binary index

The filter is applied only if the binary indices in a 3x3 window are all the same, i.e., all “0” indices or all “1” indices. Note 10x10 binary indices are obtained with a single threshold which corresponds to the 8x8 block shown in the above figure, where the shaded region represents the pixels to be filtered.

The filter coefficients used for both intra and non-intra blocks denoted by *coef(i,j),* where *i,j=-1,0,1*, are:

1	2	1
2	4	2
1	2	1

Figure F.24 — Filter mask for adaptive smoothing

Here the coefficient at the center pixel, i.e., *coef(0,0)*, corresponds to the pixel to be filtered. The filter output *flt'(i,j)* is obtained by:

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$flt'(h,v) = \{8 + \sum_{i=-1}^{1} \sum_{j=-1}^{1} coef(i,j) \cdot rec(h+i, v+j)\ \} // 16$$

F.3.2.3.2 Clipping

The maximum gray level change between the reconstructed pixel and the filtered one is limited according to the quantiser scale, i.e., QP. Let *flt(h,v)* and *flt'(h,v)* be the filtered pixel value and the pixel value before limitation, respectively.

```
if( flt'(h,v) - rec(h,v) > max_diff )
    flt(h,v) = rec(h,v) + max_diff
else if( flt'(h,v) - rec(h,v) < -max_diff )
    flt(h,v) = rec(h,v) - max_diff
else
    flt(h,v) = flt'(h,v)
```

where max_diff=QP/2 for both intra and inetr macroblocks.

F.3.3 Further issues

In order to reduce the number of computations in post-filtering, two kinds of semaphores can be defined: the blocking semaphores and the ringing semaphore. Depending on the blocking semaphores, the horizontal and the vertical deblocking filtering is applied strongly and weakly on the block boundary. If the ringing semaphore (RS) of a current block is “1”, deringing filtering is applied. For extracting the semaphores in intra-frame, when only a DC component in the 8x8 inverse quantised coefficients (IQC), the DCT coefficients after inverse quantisation, has a non-zero value, both the horizontal blocking semaphore (HBS) and the vertical blocking semaphore (VBS) of the block are set to “1”. When only the coefficients in the top row of the IQC have non-zero values, the VBS is set to “1”. When only the coefficients in the far left column of the IQC have non-zero values, the HBS is set to “1”. The RS is set to “1” if any non-zero coefficient exists in positions other than a DC component, the first horizontal AC component, and the first vertical AC comonent. Also the semaphores of the inter-frame are calculated from both the residual signal and the semaphores of the reference frame by using the motion vector.

[1] M2723 Y.L. Lee et al.

F.4 Chrominance Decimation and Interpolation Filtering for Interlaced Object Coding

The chrominance decimation filtering and the chrominance interpolation filtering are performed on the basis of the same field in case of interlaced object-based coding during the image format conversion process from 4:2:2 to 4:2:0 and from 4:2:0 to 4:2:2, respectively. In case that filter taps contain only object samples, conventional filtering methods are applied. When filter taps contain both object and background samples, the following decimation and interpolation filtering methods are applied:

Decimation filtering: The source object and background chroma samples are identified from the luminance shape information. Among the decimated chroma samples, the object chroma samples are identified by the field-based chroma subsampling method described in subclause 7.6.1.5. When filter taps contain both of object and background samples at the object boundary, only chroma sample values of object are used for obtaining its decimated chroma sample value. For example, in case of a two-tap filter, one tap is on an object sample and the other is on a background sample, the resulting chroma sample is obtained using only the object chroma sample, not counting on the background sample. In Figure F.25 (a), the source chroma sample C1 is object and C2 is background. The value of the resulting decimated chroma sample C3 is the value of C1.

Interpolation filtering: In the interpolation filtering process which is the inverse of the decimation filtering, object chroma samples are identified among the interpolated chroma samples from the luminance shape information and the values of the interpolated object chroma samples are obtained applying an interpolation filter on input decimated chroma samples. When filter taps include both of decimated object and background chroma samples, only the decimated object chroma samples are counted. For example, in case of a two-tap filter in Figure F.25 (b), two decimated chroma samples, C4 and C5, are used for obtaining the interpolated chroma samples C6 and C7.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Suppose that C6 is identified as object and C7 is identified as background from the luminance shape information and the chroma sample C4 is object and C5 is background, then the interpolated chroma sample C6 is obtained as the value of the chroma sample C4. The interpolation for C7 is not necessary because it is background.

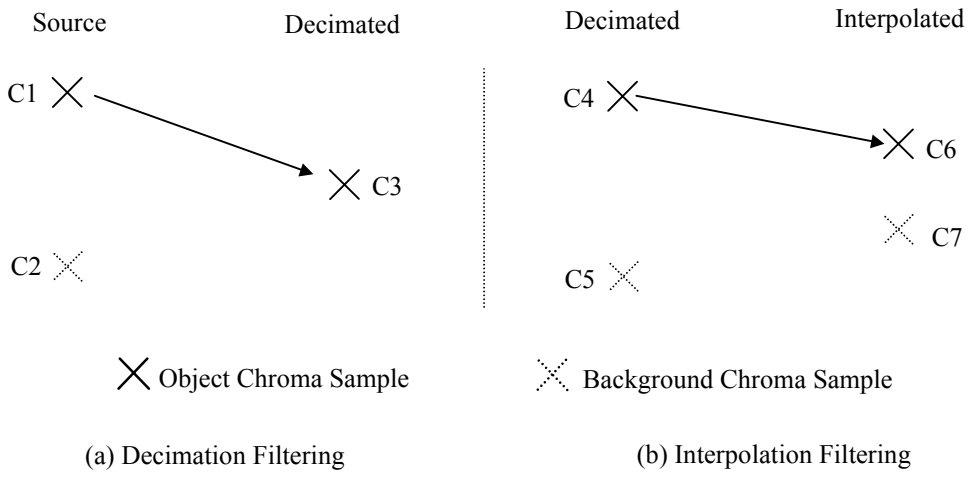


Figure F.25 — Chrominance Decimation and Interpolation Filtering at the Object Boundaries

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex G
(normative)

Profile and level indication and restrictions

This annex specifies the syntax element restrictions and permissible layer combinations.

Table G.1 — FLC table for profile_and_level_indication

Profile/Level	Code
Reserved	00000000
Simple Profile/Level 1	00000001
Simple Profile/Level 2	00000010
Simple Profile/Level 3	00000011
Reserved	00000100 – 00000111
Simple Profile/Level 0	00001000
Reserved	00001001 - 00001111
Simple Scalable Profile/Level 0	00010000
Simple Scalable Profile/Level 1	00010001
Simple Scalable Profile/Level 2	00010010
Reserved	00010011 – 00100000
Core Profile/Level 1	00100001
Core Profile/Level 2	00100010
Reserved	00100011 – 00110001
Main Profile/Level 2	00110010
Main Profile/Level 3	00110011
Main Profile/Level 4	00110100
Reserved	00110101 – 01000001
N-bit Profile/Level 2	01000010
Reserved	01000011 – 01010000
Scalable Texture Profile/Level 1	01010001
Reserved	01010010 – 01100000
Simple Face Animation Profile/Level 1	01100001
Simple Face Animation Profile/Level 2	01100010
Simple FBA Profile/Level 1	01100011
Simple FBA Profile/Level 2	01100100
Reserved	01100101 – 01110000
Basic Animated Texture Profile/Level 1	01110001
Basic Animated Texture Profile/Level 2	01110010
Reserved	01110011 – 10000000
Hybrid Profile/Level 1	10000001
Hybrid Profile/Level 2	10000010
Reserved	10000011 – 10010000
Advanced Real Time Simple Profile/Level 1	10010001
Advanced Real Time Simple Profile/Level 2	10010010
Advanced Real Time Simple Profile/Level 3	10010011

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Advanced Real Time Simple Profile/Level 4	10010100
Reserved	10010101 – 10100000
Core Scalable Profile/Level1	10100001
Core Scalable Profile/Level2	10100010
Core Scalable Profile/Level3	10100011
Reserved	10100100 – 10110000
Advanced Coding Efficiency Profile/Level 1	10110001
Advanced Coding Efficiency Profile/Level 2	10110010
Advanced Coding Efficiency Profile/Level 3	10110011
Advanced Coding Efficiency Profile/Level 4	10110100
Reserved	10110101 – 11000000
Advanced Core Profile/Level 1	11000001
Advanced Core Profile/Level 2	11000010
Reserved	11000011 – 11010000
Advanced Scalable Texture/Level1	11010001
Advanced Scalable Texture/Level2	11010010
Advanced Scalable Texture/Level3	11010011
Reserved	11010100 – 11100000
Simple Studio Profile/Level 1	11100001
Simple Studio Profile/Level 2	11100010
Simple Studio Profile/Level 3	11100011
Simple Studio Profile/Level 4	11100100
Core Studio Profile/Level 1	11100101
Core Studio Profile/Level 2	11100110
Core Studio Profile/Level 3	11100111
Core Studio Profile/Level 4	11101000
Reserved	11101001 – 11101111
Advanced Simple Profile/Level 0	11110000
Advanced Simple Profile/Level 1	11110001
Advanced Simple Profile/Level 2	11110010
Advanced Simple Profile/Level 3	11110011
Advanced Simple Profile/Level 4	11110100
Advanced Simple Profile/Level 5	11110101
Reserved	11110110
Advanced Simple Profile/Level 3b	11110111
Fine Granularity Scalable Profile/Level 0	11111000
Fine Granularity Scalable Profile/Level 1	11111001
Fine Granularity Scalable Profile/Level 2	11111010
Fine Granularity Scalable Profile/Level 3	11111011
Fine Granularity Scalable Profile/Level 4	11111100
Fine Granularity Scalable Profile/Level 5	11111101
Reserved	11111110
Reserved for Escape	11111111

Ed. note (JO): 255=ESCAPE needs to be re-defined by corrigendum due to mismatch with previous systems definition: 255=No Visual Capability required (critical in context of File Format)

-`-`-`,,`-`-`,,`,,`,`,,`-`-`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table G.2 — possible combination of two tools

Tools		BA	B-VOP	OBMC	Q	ER	SH	BS	GS	IN	NB	TS(B)	TS(E)	SS(B)	SS(E)	SP	ST	RRV	NP	GMC	QMC	SADCT	ERST	WT	SSST	OSS(B)	OSS(E)	MAC	FGS	FGST
Basic • I-VOP • P-VOP • AC/DC Prediction • 4-MV, Unrestricted MV	BA			✓	✓	✓	✓ a)	✓	✓	✓	✓	✓	✓		✓	✓ b)	×	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	×	×
B-VOP	BV			✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	×	✓	✓	✓	×	×	×	✓	✓	✓	×	×
P-VOP with OBMC (Texture)	OBMC				✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	D	×	✓	×	✓	✓	✓	×	×	×	✓	✓	✓	×	×
Method 1/Method 2 Quantisation	Q					✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	×	×
Error resilience	ER						×	✓	✓ d)	✓ e)	✓	✓	×	✓	×	×	×	✓	✓	✓	✓	✓	×	×	×	✓	×	✓ d)	×	×
Short Header	SH							×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Binary Shape (progressive)	BS								×	✓	✓	✓	✓	D	D	✓	H	×	×	✓	✓	✓	×	×	×	✓	✓	✓	×	×
Greyscale Shape	GS									✓	D	D	D	D	D	✓ f)	H	×	×	✓	✓	✓	×	×	×	D	D	✓ g)	×	×
Interlace	IN										✓	D	D	D	D	×	×	×	×	✓	✓	✓	×	×	×	D	D	✓	✓	✓
N-Bit	NB											✓ c)	✓ c)	D	D	×	×	×	×	D	D	D	×	×	×	D	D	D	×	×
Temporal Scalability (Base)	TS(B)															D	×	×	×	✓	✓	✓	×	×	×	D	D	D	×	×
Temporal Scalability (Enhancement)	TS(E)															×	×	×	×	×	D	D	×	×	×	D	D	D	×	×
Spatial Scalability (Base)	SS(B)															D	×	×	×	✓	✓	✓	×	×	×	✓	✓	D	×	×
Spatial Scalability (Enhancement)	SS(E)															×	×	×	×	×	D	D	×	×	×	✓	✓	D	×	×
Sprite	SP																×	×	×	×	×	×	×	×	×	×	×	×	×	×
Still Texture	ST																	×	×	×	×	×	✓	✓	✓	×	×	×	×	×
Reduced Resolution VOP	DRC																		✓	×	×	×	×	×	×	×	×	×	×	×
NEWPRED	NP																			×	×	×	×	×	×	×	×	×	×	×
Global Motion Compensation	GMC																				✓	✓	×	×	×	×	×	D	×	×
Quarter-pel Motion Compensation	QMC																					✓	×	×	×	×	×	D	×	×
SA-DCT	SADCT																						×	×	×	×	×	D	×	×
Error Resilience for Still Texture Coding	ERST																							✓	✓	×	×	×	×	×
Wavelet Tiling	WT																								✓	×	×	×	×	×
Scalable Arbitrary Shape for Still Texture Coding	SSST																									×	×	×	×	×

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Object Based Spatial Scalability (Base)	OSS(B)																										D	×	×
Object Based Spatial Scalability (Enhancement)	OSS(E)																										D	×	×
Multiple Auxiliary Component	MAC																											×	×
Fine Granularity Scalability	FGS																												✓
Fine Granularity Scalability Temporal	FGST																												

✓ - This combination is supported by the syntax. (Subject to further restrictions imposed by the definition of visual object types).
× – This combination is not supported by the syntax.

D – This combination is supported by the syntax but not allowed within one Visual Object(). This combination may be supported in the future, if requirements arise for the corresponding combination.

H - supported at a higher syntactic level.

a) Short Header does not support DC/AC Prediction, 4MV nor Unrestricted MV
b) Sprite does not support 4MV nor Unrestricted MV.
c) P-VOP Temporal Scalability only.
d) Grey Scale does not support Data Partitioning nor RVLC
e) Interlace does not support Data Partitioning nor RVLC
f) Grey Scale does not support Low Latency Sprites.
g) MAC is extension of grayscale shape, no real combination

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex H
(informative)

Patent statements

The International Organization for Standardization and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this part of ISO/IEC 14496 may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patents right are registered with ISO and IEC. Information may be obtained from the companies listed below.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 14496 may be the subject of patent rights other than those identified in this annex. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

H.1 Patent statements for ISO/IEC 14496 Version 1

The table summarises the formal patent statements received and indicates the parts of ISO/IEC 14496 Version 1 to which the statement applies (S: ISO/IEC 14496-1:2001, V: tools listed in Table 9-1, A: ISO/IEC 14496-3:2001, R: ISO/IEC 14496-5:2000, D: ISO/IEC 14496-6:1999). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

		S	V	A	R	D
1.	Alcatel	x	x	x	x	x
2.	AT&T					
3.	BBC	x	x	x	x	x
4.	Bosch		x	x	x	
5.	British Telecommunications	x	x	x	x	x
6.	Canon	x	x	x	x	x
7.	CCETT	x	x	x	x	x
8.	Columbia University	x	x	x	x	x
9.	Creative	x		x	x	
10.	CSELT			x		
11.	DEmoGraFX		x		x	
12.	DirecTV	x	x	x		
13.	Dolby	x	x	x	x	x
14.	EPFL	x	x	x		
15.	ETRI	x	x	x	x	x
16.	FhG	x	x	x	x	x
17.	France Telecom	x	x	x	x	x
18.	Fujitsu Limited	x	x	x	x	x
19.	GC Technology Corporation	x	x	x		
20.	General Instrument		x		x	

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

		S	V	A	R	D
21.	Hitachi	x	x	x	x	x
22.	Hyundai	x	x	x	x	x
23.	IBM	x	x		x	
24.	Institut für Rundfunktechnik	x	x	x		x
25.	InterTrust					
26.	JVC	x	x	x	x	x
27.	KDD Corporation	x	x			
28.	KPN	x	x	x	x	x
29.	LG Semicon					
30.	Lucent					
31.	Matsushita	x	x	x	x	x
32.	Microsoft	x	x	x	x	x
33.	MIT					
34.	Mitsubishi	x	x	x	x	
35.	Motorola		x		x	
36.	NEC Corporation	x	x	x	x	x
37.	NHK	x	x	x	x	x
38.	Nokia		x	x	x	
39.	NTT	x	x	x	x	x
40.	OKI	x	x	x	x	x
41.	Philips	x	x	x	x	x
42.	PictureTel Corporation		x		x	
43.	Rockwell	x	x	x	x	x
44.	Samsung	x	x	x		
45.	Sarnoff	x	x	x	x	x
46.	Scientific Atlanta	x	x	x	x	x
47.	Sharp	x	x	x	x	x
48.	Siemens	x	x	x		
49.	Sony	x	x	x	x	x
50.	Telenor	x	x	x	x	x
51.	Teltec DCU		x		x	
52.	Texas Instruments					
53.	Thomson	x	x	x		
54.	Toshiba		x			
55.	Unisearch Ltd.		x		x	
56.	Vector Vision		x			

H.2 Patent statements for the extensions provided in ISO/IEC 14496 Version 2

The table summarises the formal patent statements received and indicates the parts of the extensions provided in ISO/IEC 14496 Version 2 to which the statement applies (S: ISO/IEC 14496-1:2001, V: tools listed in Figure 7-83 and not listed in Table 9-1, A: ISO/IEC 14496-3:2001, R: ISO/IEC 14496-5:2000/Amd.1:2000, D: ISO/IEC 14496-6:1999/Amd.1:2000). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

		S	V	A	R	D
1.	Apple	x			x	
2.	AT&T	x	x	x		
3.	British Telecommunications					
4.	Bosch	x	x	x	x	x
5.	CCETT	x	x	x	x	x
6.	Columbia Innovation Enterprise	x	x	x	x	x
7.	DemoGraFX	x	x	x	x	x
8.	DirecTV	x	x	x		
9.	Dolby	x	x	x	x	x
10.	EPFL	x	x		x	
11.	France Telecom	x	x	x	x	x
12.	Fraunhofer			x	x	
13.	Fujitsu		x		x	
14.	Hitachi	x	x	x	x	x
15.	Hyundai	x	x	x	x	x
16.	IBM	x	x	x	x	x
17.	Intertrust					
18.	JVC	x	x	x	x	x
19.	KPN		x			
20.	Lucent					
21.	Matsushita Electric Industrial Co., Ltd.	x	x	x	x	x
22.	Microsoft	x	x	x	x	x
23.	Mitsubishi	x	x	x	x	x
24.	NEC	x	x	x	x	x
25.	NHK	x	x	x	x	x
26.	Nokia	x		x	x	
27.	NTT		x			
28.	NTT Mobile Communication Networks			x		
29.	OKI	x	x	x	x	x
30.	Optibase	x			x	
31.	Philips					
32.	Samsung	x	x	x	x	
33.	Sarnoff	x	x	x	x	x
34.	Sharp	x	x	x	x	x
35.	Siemens	x	x	x	x	x
36.	Sony	x	x	x	x	x
37.	Sun	x				
38.	Thomson	x	x	x	x	x
39.	Toshiba		x			

H.3 Patent statements for the extensions provided in ISO/IEC 14496-2:2001/AMD1

The table summarises the formal patent statements received and indicates the extensions provided in ISO/IEC 14496-2:2001/AMD1 to which the statement applies (V: tools listed in Table 9-3 and not listed in Table 9-2 and not listed in Table 9-1, R: ISO/IEC 14496-5:2000/AMD2). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

			V		R	
40.	Bosch		x		x	
41.	NHK		x		x	
42.	Sony		x		x	

H.4 Patent statements for the extensions provided in ISO/IEC 14496-2/AMD2

The table summarises the formal patent statements received and indicates the extensions provided in ISO/IEC 14496-2:2001/AMD2 to which the statement applies (V: tools listed in Table 9-4 and not listed in Table 7-36 and not listed in Table 9-2 and not listed in Table 9-1, R: ISO/IEC 14496-5:2000/AMD1). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

			V		R	
1.	Bosch		x		x	
2.	Microsoft		x		x	
3.	Optivision		x		x	
4.	Philips		x		x	
5.	Vector Vision		x		x	
6.	WebCast Technologies		x		x	

H.5 Patent statements for ISO/IEC 14496-2:2001/AMD3

The table summarises the formal patent statements received and indicates the parts of ISO/IEC 14496-2:2001/AMD3 to which the statement applies (V: visual tools in this amendment, R: ISO/IEC 14496-5:2001). The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

		V	R
57.	PacketVideo	x	x
58.	Envivio	x	x
59.	Philips	x	x

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex I
(informative)

Encoder Complexity Reduction Based on Intelligent Pre-Quantisation

I.1 Introduction

This subclause describes an encoder complexity reduction scheme. The proposed algorithm is aimed at reducing computations by a significant amount at the encoder. It is based on extracting a low cost feature (SAE) from the motion compensated residual blocks, and using this feature and an intelligently derived threshold value to detect "zero blocks" (blocks have all zero quantised DCT coefficients). Once a "zero block" is detected, we can skip DCT, quantisation, and corresponding IDCT and inverse quantisation for this block to reduce the encoder computation.

Most of the existing video coding standards (MPEG-1, MPEG-2, MPEG-4, H.261 and H.263) are based on motion estimation, DCT, quantisation, and entropy coding. The core of these algorithms is a differential loop where a displaced frame difference is calculated based on the motion of the current frame from the immediate past frame. This displaced frame difference is then quantised and entropy coded via a variable length code. In the end these variable length codes, along with the codes which contain the header information conforming to the syntax of a particular algorithm, are transmitted over the channel. Motion estimation is the process by which we calculate the displacement of one frame in a sequence with respect to the previous frame. By calculating the displacement on a block basis, we can compute a displaced frame difference that is easier to code, thereby reducing the temporal redundancy. In the next step, DCT of individual blocks (8x8 pixels) helps in removing the spatial redundancy by bringing down the most relevant information into the lower most coefficients in the DCT domain. After having obtained a block of DCT coefficients that contain the energy of the displaced blocks, quantisation of these blocks is done. Quantisation is a uniform quantisation with step size varying from 2 to 62. It is implemented as a division, or as a table look up operation for a fixed-point implementation. Lastly, having obtained the quantised coefficients, entropy coding is done to further reduce the overall coding rate. Traditionally most video coding standards use Huffman coding for entropy coding.

In terms of computations motion estimation is the costliest module. Typically, it accounts for more than half of the MIPs at the encoder. For this reason, most of the video encoding solutions prefers to do motion estimation either by some dedicated hardware, or by some fast sub-optimal software scheme. DCT and IDCT, which is needed in the encoder for reconstructing the quantised frame for motion estimation, comprise the next most costly part of the encoder. Then follows quantisation and Huffman encoding. In very low bit rate applications, after the quantisation of the motion compensated DCT blocks, many of the blocks have all of their coefficients as zero. It is observed that a majority of the blocks are zero after quantisation. This means that if we can detect a priori which blocks are being zeroed out after quantisation, we can forego DCT, quantisation, inverse quantisation and IDCT for those blocks. This would translate to a great amount of savings in computations at the encoder. In the following, we outline a method to achieve this goal with no loss in the perceived quality of the coded sequence.

I.2 Feature Selection and Pre-quantisation

We begin with identifying some key features of the motion compensated blocks that will help in classifying them into zero and non-zero blocks. A good feature should allow us to classify the blocks very accurately and at the same time it should be relatively easy to compute too. Using a complicated classification scheme defeats our main goal of reducing the overall computations. We consider several features and rank them in the order of their performance.

Most of the features we select are easy to compute and are based on first or second ordered moments of the motion compensated blocks. We consider the following features:

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SAE: mean absolute error

$$\mathrm{SAE} = \sum_{i=1}^{8}\sum_{j=1}^{8} |f(i,j)| \text{ , where } f(i,j) \text{ is the pel value at } (i,j) \text{ - th location}$$

MSE: mean square error

$$\mathrm{MSE} = \sum_{i=1}^{8}\sum_{j=1}^{8} (f(i,j))^2 \text{ , where } f(i,j) \text{ is the pel value at } (i,j) \text{ - th location}$$

MAD: mean absolute difference

$$\mathrm{MAD} = \sum_{i=1}^{8}\sum_{j=1}^{8} |f(i,j) - \mu| \text{ where } f(i,j) \text{ is the pel value at } (i,j) \text{ - th location } \& \mu \text{ is the mean of the block}$$

VAR: block variance

$$\mathrm{VAR} = \sum_{i=1}^{8}\sum_{j=1}^{8} (f(i,j) - \mu)^2 \text{ , where } f(i,j) \text{is the pel value at } (i,j) \text{ - th location } \& \mu \text{ is the mean of the block}$$

MAX: maximum value

$$\mathrm{MAX} = \max(f(i,j)) \;\; i,j \in (1,8)$$

We use each of the above features in classifying the motion compensated blocks as zero or one. A zero block is defined as a block with all zero quantised DCT coefficients and a one block as a block with at least one non-zero quantised DCT coefficient. For doing the classification, we select one of the above features and use a corresponding (experimentally determined) threshold to decide whether or not this block should be classified as zero. Each time we classify a block as zero, we would be zeroing out the block to begin with and would not be doing DCT and quantisation, and consequently IDCT and inverse quantisation. This translates to a lot of savings in computations. Next we define some terminology which would help us in ranking the performance of different features and in quantifying the degradation in performance while using any such feature based classification scheme.

As is the case with any multiple hypotheses problem, there are two types of errors associated with this feature based classification scheme. Each time we use a feature to do the above classification, there are instances where we classify a block as zero, when it should have been coded instead (one). We call this error as false alarm (FA). The second type of error we encounter in such a classification scheme is when we classify a block to be one when in reality it should have been zero. We call this as misdetection (MD). We formally define a quantitative measure of false alarm and misdetection as follows:

$$\mathrm{FA}_{rate} = \frac{\mathrm{FA}}{\mathrm{N1}} \text{ , } \mathrm{MD}_{\mathrm{rate}} = \frac{\mathrm{MD}}{\mathrm{N0}} \text{ ,}$$

where FA is the number of blocks classified as zero when they actually should have been one, MD is the number of blocks classified as one when they actually should have been zero, N1 is the number of true zero blocks, and N0 is the number of true one blocks. We compare the effectiveness of different features by plotting false alarm (FA) vs. misdetection (MD) curves resulting from using the above features in coding a video sequence. Ideally we would want a feature to give us a low false alarm and a low misdetection.

From our experiments we conclude that SAE and MSE have the best performance (lowest false alaram rate and lowest misdetection rate), MAX has the worst performance (highest curve), and MAD, VAR are in between. Since SAE is computationally less expensive to calculate than MSE, we will use SAE as our major feature in classifying the blocks. Also, since MAX is very easy to compute, and it also promises to catch some edge information in a block, we will also investigate using it further.

--`,,`-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

I.3 Model Verification and Threshold Setting

We apply the classification schemes based on above features on a wide range of video sequences. Since motion estimation acts like a normalization process, the statistics of residue blocks is expected to be similar. We verify this fact by encoding several typical sequences. We observe that the curves for misdetection rate and false alarm rate are similar for all sequences. In other words, the feature classification scheme is quite robust to the variations in the input sequence.

Based on experiments, we find SAE gives best result in terms of false alarm and misdetection rate. Here we will find an optimal threshold that results in negligible number of false alarm blocks and at the same time small number of misdetection blocks. Negligible false alarm rate means only very few true one blocks will be misclassified as zero blocks, therefore, the quality degradation will be totally insignificant. Small misdetection rate means most true zero blocks will be classified as zero blocks, and these blocks will skip DCT and quantisation, therefore, computational complexity will be reduced. Here we first calculate the upper bound of SAE that will guarantee zero false alarm rate.

The forward DCT is:

$$F(u,v) = \frac{1}{4} C(u)\, C(v) \sum_{i=0}^{7} \sum_{j=0}^{7} \cos((2i+1)u\frac{\pi}{16}) \cos((2j+1)v\frac{\pi}{16}) f(i,j)$$

$$=> |F(u,v)| \le \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} |f(i,j)| = 1/4\, SAE, \text{ for } u, v \neq 0$$

$$= 1/8\, SAE, \text{ for } u, v = 0$$

I.3.1 H.263 Quantiser Scale

When using the H.263 quantiser scale, the quantisation for INTER mode blocks AC/DC coefficients can be expressed as:

$$coeff(u,v) = sign(F(u,v)) * \frac{|F(u,v)| - QP/2}{2*QP}$$

In this case, to guarantee all coeff(u, v)=0, we have

$$|F(u,v)| - QP/2 < 2*QP$$
$$|F(u,v)| < 5/2 * QP$$

From above equations we can guarantee that the DC componet of the block would be zero if the following condition is satisfied:

$$\frac{1}{8} SAE < 5/2 * QP$$
$$SAE < 20 * QP$$

I.3.2 MPEG-4 Quantiser Scale

When using the MPEG-4 quantiser scale with the MPEG-4 non-intra quantisation matrix, the quantisation for INTER mode blocks AC/DC coefficients can be expressed as:

$$coeff(u,v) = sign(F(u,v)) * \frac{(16 * |F(u,v)|)//w[u][v]}{2 * QP},$$

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

where w is the non-intra quantisation matrix. In this case, to guarantee all coeff(u, v)=0, we have

$$|F(u,v)| < 2*QP$$

From above equations we can guarantee that the DC componet of the block would be zero if the following condition is satisfied:

$$\frac{1}{8}SAE < 2*QP$$
$$SAE < 16*QP$$

Since the DC component is the most important and more likely to have non-zero value if a block has any non-zero components, this threshold value guarantees no quality degradation when applying the pre-quantisation scheme and receiving coding complexity reduction.

To conclude, the threshold is set to 20*QP when H263 quantiser scale is used, and 16*QP when MPEG-4 quantiser scale with default quantisation matrix is used. These thresholds will guarantee accurate prediction of zero coefficients both for AC as well as the DC coefficients.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex J
(normative)

View dependent object scalability

J.1 Introduction

Coding of View-Dependent Scalability (VDS) parameters for texture can provide for efficient incremental decoding of 3D images (e.g. 2D texture mapped onto a gridded 3D mesh such as terrain). Corresponding tools from ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496 are used in conjunction with downstream and upstream channels of a decoding terminal. The combined capabilities provide the means for an encoder to react to a stream of viewpoint information received from a terminal. The encoder transmits a series of coded textures optimized for the viewing conditions which can be applied in the rendering of textured 3D meshes by the receiving terminal. Each encoded view-dependent texture (initial texture and incremental updates) typically corresponds to a specific 3D view in the user's viewpoint that is first transmitted from the receiving terminal.

A tool defined in ISO/IEC 14496-1 transmits 3D viewpoint parameters in the upstream channel back to the encoder. The encoder's response is a frequency-selective, view-dependent update of DCT coefficients for the 2D texture (based upon view-dependent projection of the 2D texture in 3D) back to the receiving terminal, along the downstream channel, for decoding by a Visual DCT tool at the receiving terminal. This bilateral communication supports interactive server-based refinement of texture for low-bandwidth transmissions to a decoding terminal that renders the texture in 3D for a user controlling the viewpoint movement. A gain in texture transmission efficiency is traded for longer closed-loop latency in the rendering of the textures in 3D. The terminal coordinates inbound texture updates with local 3D renderings, accounting for network delays so that texture cached in the terminal matches each rendered 3D view.

A method to obtain an optimal coding of 3D data is to take into account the viewing position in order to transmit only the most visible information. This approach reduces greatly the transmission delay, in comparison to transmitting all scene texture that might be viewable in 3D from the encoding database server to the decoder. At a given time, only the most important information is sent, depending on object geometry and viewpoint displacement. This technique allows the data to be streamed across a network, given that a upstream channel is available for sending the new viewing conditions to the remote database. This principle is applied to the texture data to be mapped on a 3D grid mesh. The mesh is first downloaded into the memory of the decoder using the appropriate BIFS node, and then the DCT coefficients of the texture image are updated by taking into account the viewing parameters, i.e. the field of view, the distance and the direction to the viewpoint.

J.2 Decoding Process of a View-Dependent Object

This clause explains the process for decoding the texture data using the VDS parameters. In order to determine which of the DCT coefficients are to be updated, a "mask", which is a simple binary image, shall be computed. The first step is to determine the viewing parameters obtained from the texture-mesh composition procedure that drives 3D rendering in the user's decoding terminal. These parameters are used to construct the DCT mask corresponding to the first viewpoint of the session (VD mask). This mask is then updated with differential masks, built with the new viewing parameters that allow the texture image to be streamed. The bitstream syntax for view parameters and incremental transmission of DCT coefficients is given elsewhere in ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496.

J.2.1 General Decoding Scheme

The following subclauses outline the overall process for the decoder and encoder to accomplish the VDS functionalities.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

J.2.1.1 View-dependent parameters computation

The VDS parameters (α and β angles, distance d for each cell) shall be computed using the geometrical parameters (Mesh, Viewpoint, Aimpoint, Rendering window). These parameters shall be computed for each cell of the grid mesh.

J.2.1.2 VD mask computation

For each 8x8 block of texture elements within a 3D mesh cell, the locations of the visible DCT coefficients inside the DCT block shall be computed using α and β angles, and the distance d defined for each cell relative to the viewpoint. The result shall be put in a binary mask image.

J.2.1.3 Differential mask computation

With the knowledge of which DCT coefficients have already been received (Binary mask buffered image) and which DCT coefficients are necessary for the current viewing conditions (Binary VD mask image), the new DCT coefficients shall be determined (Binary Differential mask image) as described in subclause J.2.4.

J.2.1.4 DCT coefficients decoding

The Video Intra bitstream, in the downstream channel, shall be decoded by the receiver terminal to obtain the DCT coefficients (DCT image). The decoding procedure is described in subclause J.2.5.

J.2.1.5 Texture update

The current DCT buffer in the receiver terminal shall be updated according to the Differential mask, using the received DCT image. The new received DCT coefficients shall be added to the buffered DCT image.

J.2.1.6 IDCT

The Inverse DCT of the updated DCT image shall computed, as specified in subclause J.2.7, to obtain the final texture.

J.2.1.7 Rendering

The texture is mapped onto the 3D mesh and the rendering of the scene is done, taking into account the mesh and the viewing conditions. This part of the procedure is outside the scope of this part of ISO/IEC 14496.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

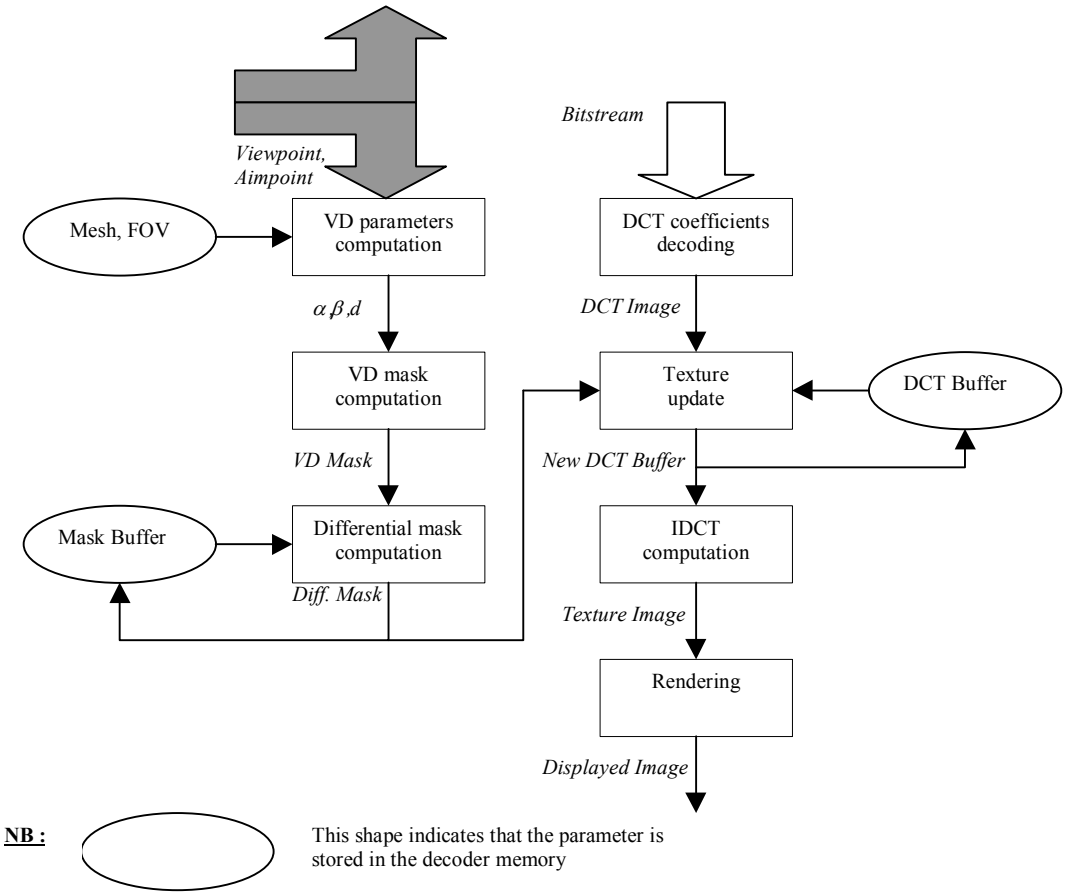


Figure J.1— General Decoding Scheme of a View-Dependent Object

J.2.2 Computation of the View-Dependent Scalability parameters

The VDS parameters shall be computed for each cell of the grid mesh. The mesh may either be a quadrilateral or a triangular mesh. The number of cells in each dimension shall be equal to the texture size divided by 8.

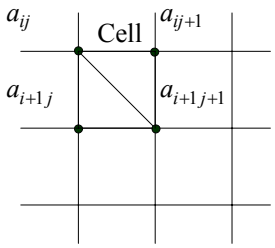


Figure J.2 — Mesh cell

J.2.2.1 Distance criterion:

$$Rd = \frac{1}{u}$$

u is the distance between viewpoint and Cell center: $u = \|\vec{v} - \vec{c}\|$ with $\vec{c} = \frac{1}{4}(\vec{a}_{ij} + \vec{a}_{i+1j} + \vec{a}_{ij+1} + \vec{a}_{i+1j+1})$ and $\vec{v}$ is the viewpoint vector.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

J.2.2.2 Rendering criterion:

$$R_r = \frac{p}{q}$$

p is the distance between viewpoint and projection plane normalized to window width. *p* may be computed using:

$$p = \frac{1}{2\tan(FOV/2)}$$

$$p = \frac{1}{2\tan(FOV/2)}, \text{FOV is the Field Of View}$$

where FOV is the Field of View specified in radians, and q = <TextureWidth>/<WindowWidth> where the texture width is the width of the full texture (1024 for instance) and the WindowWidth is the width of the rendering window.

J.2.2.3 Orientation criteria:

$$Ra = \cos(\alpha)$$
$$Rb = \cos(\beta)$$

The angle between the aiming direction and the normal of the current cell center shall be projected into two planes. These two planes are spans of normal vector $\vec{n}$ of the cell and the cell edges in x and y directions, respectively. Then the angles (α, β) between projected vectors and the normal $\vec{n}$ shall be calculated, respectively.

The angle β is specified as the projection of the angle between $\vec{n}$, the normal of the quad cell, and $\vec{u}$, the aiming direction, onto the plane Π_x that passes through $\vec{g}_x$ and is parallel to $\vec{n}$. Similarly, the angle α is specified as the projection of the same angle onto the plane Π_y that passes through $\vec{g}_y$ and its parallel to $\vec{n}$.

This is illustrated in Figure J.3

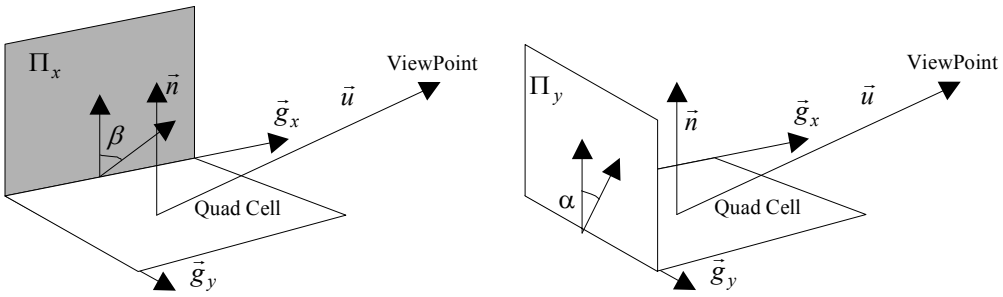


Figure J.3 — Definition of α and β angles

J.2.2.4 Cropping criterion:

Cells that are out of the field of view shall not be transmitted/received: that is, at least one of the 4 vertices which define the cell should all be inside the horizontal and vertical Field Of View (FOV).

The horizontal FOV shall be deduced from the vertical FOV using the screen geometry. The vertical FOV is equal to the FOV. Then the following shall be calculated

$HFOV = A\tan(\frac{w}{h}\cdot\tan(FOV/2))$ where *w* and *h* are the width and height, respectively, of the rendered image.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

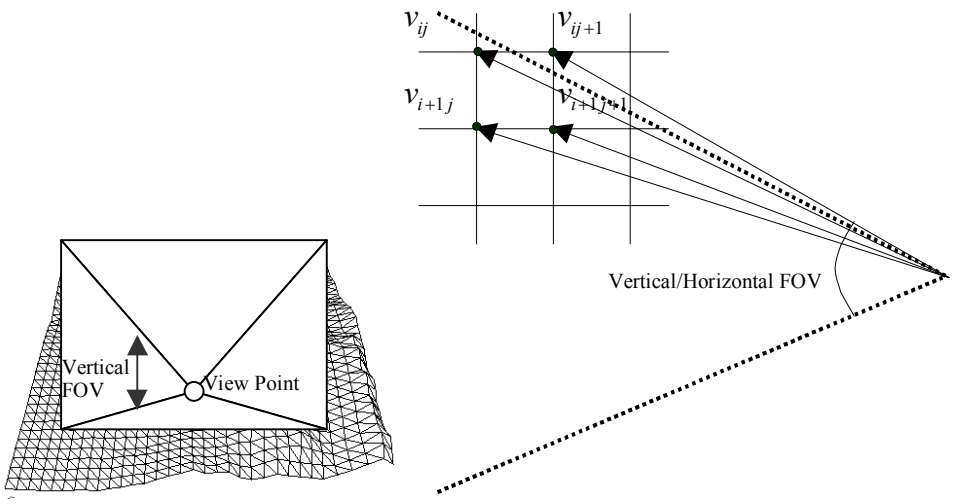


Figure J.4 — Definition of Out of Field of View cells

J.2.3 VD mask computation

The VD mask is a binary image of the same size as the texture image. Each value in the mask shall indicate if the corresponding DCT coefficient is needed (1) or not (0), given the VDS parameters.

For each cell, the following rules shall be applied to fill the corresponding 8x8 block of the VD mask:

- **Use of cropping criterion**: If all the vertices of the cell are out of the field of view, the corresponding 8x8 block of the mask image shall be set to 0.
- **Use of rendering, distance, tilting and rotation criteria**: For each 8x8 block of the mask (corresponding to a quad cell), the 4 criteria mentioned above shall be computed. Two values of the rotation and tilting criteria shall be obtained for a quad cell, but only the higher value of each criterion shall be kept.

Two thresholds, T_X and T_Y, shall be calculated as the product of the three VDS parameters *Rr, Rd, Rb,* and *Rr, Rd, Ra*, respectively, and the value 8. The results shall be bounded to 8. This procedure may be indicated symbolically as follows

$$Tx = Min\left(8, 8 \cdot R_r \cdot R_d \cdot R_b\right)$$
$$Ty = Min\left(8, 8 \cdot R_r \cdot R_d \cdot R_a\right)$$

The flag (i,j) of the 8x8 block corresponding to the current cell shall be set to 1 if i < T_X and j < T_Y. The flag shall be set to 0 in all other cases, as illustrated in the figure below.

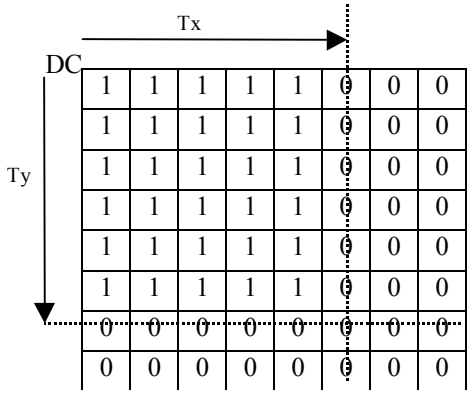


Figure J.5 — VD mask of an 8x8 block using VD parameters

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

J.2.4 Differential mask computation

Once the first image has been received using the previously described filter, less data is necessary to update the texture data for the following frames. (assuming there is a correlation between viewpoint positions). Since this computation is exactly the same for each flag of each cell, it shall be performed directly on the full mask images and not on a cell by cell basis.

If the coefficient has not been already transmitted (buffer mask set to 0) and is needed according to VDS visibility criteria (VD mask set to 1), then the corresponding pixel of the differential mask shall be set to 1. This implies that the texture shall be updated, according to the procedure described in subclause J.2.6.

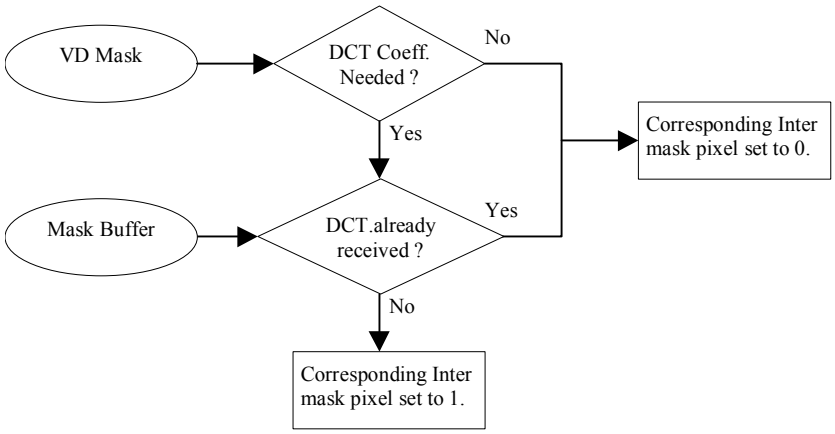


Figure J.6 — Differential mask computation scheme

J.2.5 DCT coefficients decoding

The DCT coefficients shall be decoded using the Video Intra mode, Separated Texture/Motion mode as described in the subclause 7.4.

J.2.6 Texture update

The Differential mask image shall be used to select which DCT coefficients of the buffered texture should be updated using the decoded DCT coefficients.

- Y component

If the Differential mask is set to 0, the corresponding DCT value of the buffer shall be left unchanged, otherwise the value shall be updated with the previously decoded DCT coefficient.

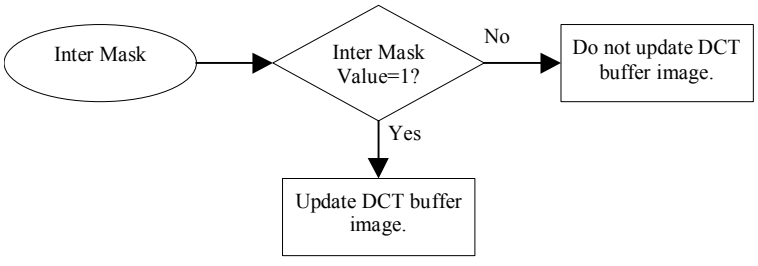


Figure J.7 — Texture update scheme

- U and V component

The texture is coded in 4:2:0 format, as specified in subclause 6.1.3.6, which shall imply that for each chrominance DCT coefficient, 4 Differential mask flags shall be available. The chrominance coefficients shall be received/transmitted if at least 1 of these 4 flags is set to 1.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

J.2.7 IDCT

The IDCT and de-quantisation shall be performed using the same process as in the Video Intra mode as described in subclause 7.4.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex K
(normative)

Decoder Configuration Information

K.1 Introduction

This annex identifies the syntax elements defined in the main body of this part of ISO/IEC 14496 that describe the configuration of the visual decoder. This Annex amplifies the information provided in subclause 6.2.1. These elements shall be processed differently if this part of the specification is used jointly with the Systems part, ISO/IEC 14496-1:2001. Instead of conveying the configuration at the beginning of a visual elementary bitstream, it shall be conveyed as part of an Elementary Stream Descriptor that is itself included in an Object Descriptor describing the visual object. This descriptor framework is specified in ISO/IEC 14496-1:2001.

K.2 Description of the set up of a visual decoder (informative)

The process of accessing ISO/IEC 14496 content is specified in ISO/IEC 14496-1:2001. It is summarized here and visualized in Figure L.1 to outline the processing of decoder configuration information in an ISO/IEC 14496 terminal. This description assumes that all elementary streams are accessible and solely the problem of identifying them is to be solved.

The content access procedure starts from an Initial Object Descriptor that may be made available through means that are not defined in this part of ISO/IEC 14496, like a http or ftp link on an HTML page, a descriptor in an ISO/IEC 13818-1 Transport Stream, or some H.245 signaling, etc. Note that this may need standardisation by the responsible group.

This Initial Object Descriptor contains pointers at least to a scene description stream and an object descriptor stream. The scene description stream conveys the time variant spatiotemporal layout of the scene. For each streaming media object incorporated in the scene, an Object Descriptor exists that describes the set of streams associated to this media object. The set of Object Descriptors is conveyed in a separate stream, in order to distinguish scene description from the description of the streaming resources.

Both the scene description stream and the object descriptor stream allow for time stamped updates of the scene description and the object descriptors, respectively. Due to the time stamps it is always known at which point in time, or from which point in time onwards a data item, called Access Unit, is valid.

The Object Descriptor associated to a given visual object is identified by its ObjectDescriptor_ID. Each visual object may require more than one elementary stream (ES) that convey its coded representation, especially if any form of scaleability is used. Each of these streams is described by an ES_Descriptor. This description contains a unique label for the stream, the ES_Id, and, among others, the DecoderSpecificInfo structure that is of concern for the purpose of this Annex.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

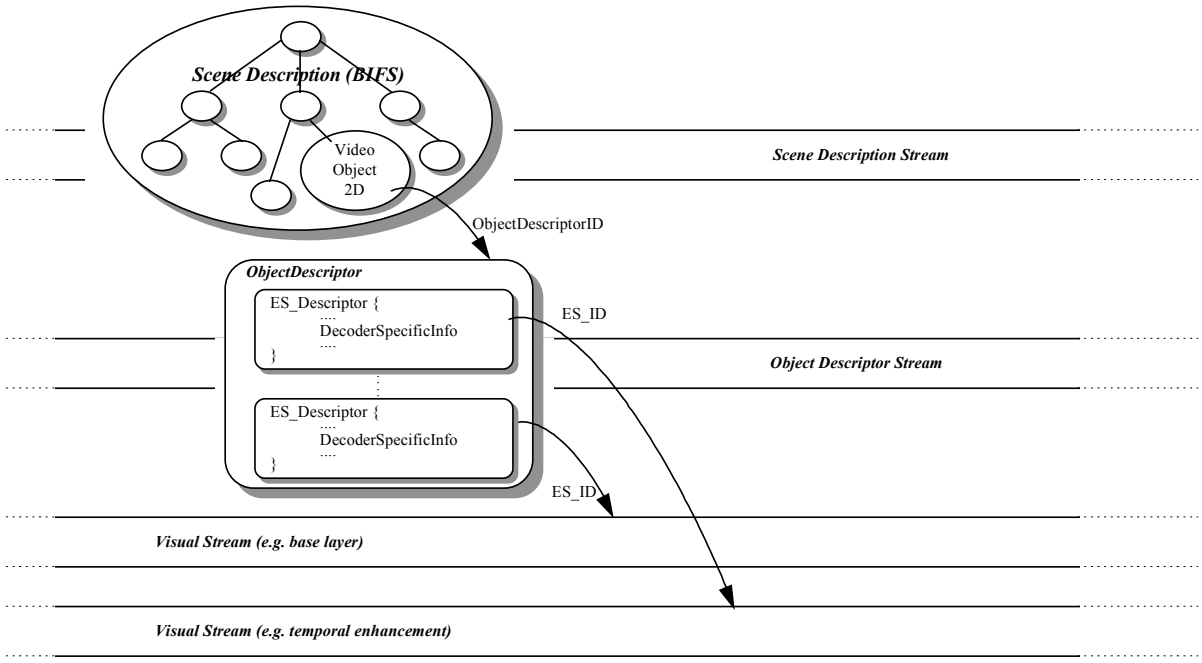


Figure K.1 — Visual decoder setup

K.2.1 Processing of decoder configuration information

After the retrieval of the Object Descriptor for a media object a decoder for the visual stream(s) is instantiated, connected to the stream(s) and initialised with the data found in ES_Descriptor.DecConfigDescr.DecSpecificInfo for each stream. Subsequently, in a random access scenario, the decoder is expected to search forward to the next random access point, while in a client-server or local storage scenario, data in the visual stream may already be aligned in a way that the first data arriving in the visual stream corresponds to a random access point.

The difference between a visual-only scenario, as specified in the main body of this part of ISO/IEC 14496, and an integrated application using both ISO/IEC 14496-1:2001 and this part of ISO/IEC 14496 is visualized in a figure. Figure K.2 shows the Visual-only approach with configuration information at the beginning of a bitstream and optionally repeated thereafter to enable random access. Figure K.3 with the integrated Systems and Visual approach shows the plain bitstreams with the configuration information extracted into the object descriptors. In this case the object descriptors will be repeated if random access is desired. The Access Units shown in the figure correspond to VOPs in the case of visual media streams.

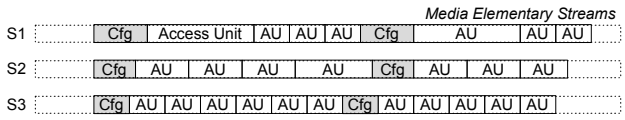


Figure K.2 — Visual-only scenario

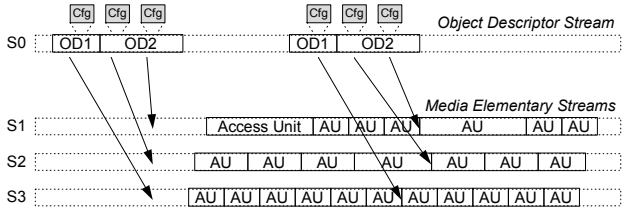


Figure K.3 — Integrated Systems and Visual approach

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

K.3 Specification of decoder configuration information

The decoder configuration information for a visual elementary stream is given by a concatenation of those syntax elements that precede the actual encoded data, i. e., that form the ‚stream header' according to the syntax specification in subclause 6.2. Those syntax elements are identified separately for each type of visual object in the subsequent subclauses. The syntax elements that are conveyed as decoder configuration information shall not be present in the visual elementary stream itself. Furthermore, the syntax definition in subclause 6.2 is constrained to clarify that an elementary stream may not contain a concatenation of, for example, multiple VisualObject() structures or multiple VideoObjectLayer() structures.

The decoder configuration information shall be conveyed in the DecConfigDescr.DecSpecificInfo in field of the respective ES_Descriptor and passed to the decoder before any data of the visual elementary stream itself.

VisualObjectSequence() is transmitted explicitly in each DecConfigDescr.DecSpecificInfo of an Object Descriptor.

Identification and priority of objects is signaled for all kinds of audiovisual objects in the ES_Descriptor. This signalling overrides the value of VisualObject.is_visual_object_identifier and VideoObjectLayer.is_object_layer_identifier.

VisualObject() carries information describing all the elementary streams associated with an object and is carried in a container associated with all the layers of this object.

VideoObjectLayer() carries information describing a single elementary stream, i.e. a single-layer object or one layer of a multi-layer object. This information is carried in a container associated with the corresponding elementary stream.

K.3.1 VideoObject

The decoder configuration information for a visual object of type VideoObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

- including all syntax elements of one VideoObject()
- including all syntax elements of one VideoObjectLayer() excluding the trailing Group_of_VideoObjectPlane() and VideoObjectPlane() that convey the coded data.

The elementary stream consists of the Group_of_VideoObjectPlane(), VideoObjectPlane(), video_plane_with_short_header and all lower layers.

K.3.2 StillTextureObject

The decoder configuration information for a visual object of type StillTextureObject consists of the following elements

- All syntax elements of VisualObjectSequence()
  - including all syntax elements of one VisualObject()
  - including all syntax elements of one StillTextureObject() up to but not including the first call to wavelet_dc_decode() in case of visual_object_verid == “0001”.
  - including all syntax elements of one StillTextureObject() up to but not including the first call to shape_object_decoding() in case of visual_object_verid != “0001” and tiling_disable==“1” and texture_object_layer_shape == “01“
  - Including all syntax elements of one StillTextureObject() up to but not including the first texture_tile_start_code in case of visual_object_verid != “0001” and tiling_disable==“0“.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

K.3.3 MeshObject

The decoder configuration information for a visual object of type MeshObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

K.3.4 FaceObject

The decoder configuration information for a visual object of type FaceObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

K.3.5 3DMeshObject

The decoder configuration information for a visual object of type 3DMeshObject consists of the following elements

- All syntax elements of VisualObjectSequence()

including all syntax elements of one VisualObject()

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex L
(informative)

Rate control

L.1 Frame Rate Control

L.1.1 Introduction

Rate control and buffer regulation is an important issue for both VBR and CBR applications. In the case of VBR encoding, the rate controller attempts to achieve optimum quality for a given target rate. In the case of CBR encoding and real-time application, the rate control scheme has to satisfy the low-latency and VBV buffer constraints. In addition, the rate control scheme has to be applicable to a wide variety of sequences and bit rates.

The scalable rate control (SRC) scheme is designed to meet both VBR without delay constraints and CBR with low-latency and buffer constraints. The SRC scheme is scalable for various bit rates (e.g. 10kbps to 1Mbps), various spatial resolutions (e.g. qcif to cif) and various temporal resolutions (e.g. 7.5fps to 30fps) and various coders (e.g. DCT and wavelet). The technique can handle I, P, and B pictures. The SRC is for single VO, and extensions to M-VOs and other improvements are being addressed in Clause L.2 The current description in Clause L.1 only handles I and P pictures.

L.1.2 Description

The SRC scheme assumes that the encoder rate distortion function can be modeled as

R=X1*S*Q**(-1)+X2*S*Q**(-2)

The encoding bit count is denoted as R. The encoding complexity which is mean absolute difference (MAD) is denoted as S. The quantiser scale is denoted as Q. The modeling parameters are denoted as X1 and X2. Because of the generality of the assumption, the scalable rate control scheme is applicable to a variety of bit rates, spatial resolutions, temporal resolutions, buffer constraints and types of coders.

There are four steps in the SRC scheme:

1) Initialization
   - X1 and X2 are the first and second order coefficients.
2) Computation of the target bit rate before encoding.
   - The target bit rate is computed based on the bits available and the last encoded frame bits. If the last frame is complex and uses excessive bits, more bits should be assigned to this frame. However, there are fewer bits left for encoding. Thus, fewer bits can be assigned to this frame. A weighted average reflects a compromise of these two factors.
   - A lower bound of target rate (R/30) is used so that minimal quality is guaranteed.
   - The target rate is adjusted according to the buffer status to prevent both overflow and underflow.
3) Computation of the quantiser scales (QP) before encoding.
   - QP is solved based on the model parameters X1 and X2.
   - QP is clipped between 1 and 31.
   - QP is limited to vary within 25 percent of the previous QP to maintain a VBR quality.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4) After encoding model parameters is updated based on the encoding results of the current frame.

- The rate distortion model is updated based on the encoding results of the current frame. The bits used for the header and the motion vectors are deducted since they are not related to QP.
- The data points are selected using a window whose size depends on the change in complexity. If the complexity changes significantly, a smaller window with more recent data points is used.
- The model is calibrated again by rejecting the outlier data points. The rejection criteria are that data point is discarded when the prediction error is more than one standard deviation.
- The next frame is skipped if the current buffer status is above 80 percents.

The algorithm is described as follows:

Step 1: Initialization after the first frame is coded

Rs: bit rate for the sequence (or segment). /* e.g., 24000 bits/sec */
Rf: bits used for the first frame. /* e.g., 10000 bits */
Rc: bits used for the current frame. It is the bit count obtained after encoding.
Rp: bits to be removed from the buffer per picture.
Ts: number of seconds for the sequence (or segment). /* e.g., 10 sec */
Ec: mean absolute difference for the current frame after motion compensation.
If the macroblock is intra coded, the original spatial pixel values are summed.
Qc: quantisation level used for the current frame.
Nr: number of P frames remaining for encoding.
Ns: distance between encoded frames. /* e.g., 4 for 7.5 fps */
Rr: number of bits remaining for encoding this sequence (or segment).
T: target bit to be used for the current frame.
S: number of bits used for encoding the previous frame.
Hc: header and motion vector bits used in the current frame. It includes all the information except to the residual information.
Hp: header and motion vector bits used in the previous frame. It includes all the information except to the residual information.
Ql: quantisation level used in the previous frame
Bs; buffer size e.g., R/2
B; current buffer level e.g., R/4 - start from the middle of the buffer

```
Rr = Ts*Rs-Rf;          /* total number of bits available for this segment */
Rp = Rr/Nr;             /* the average bits to be removed from the buffer */
if(first frame){
    Qc=15;
    goto step 4
}
```

Step 2: Target Bit Calculation (Before Encoding the Current Frame)

```
T=Max(Rs/30,Rr/Nr*0.95+S*0.05);/*each frame is assigned a minimum of R/30 */
T=T*(B+2*(Bs-B))/(2*B+(Bs-B)); /* increase if less than half */
                    /* decrease if more than half, don't change if half */
if (B+T > 0.9*Bs)
    T = Max(Rs/30, 0.9*Bs-B);   /* to avoid overflow*/
else if (B-Rp+T < 0.1*Bs)
    T = Rp-B+0.1*Bs;        /* to avoid underflow*/
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 3: Quantisation Level Calculation (Before Encoding the Current Frame)

```
T= Max(Rp/3+Hp, T);

if ((X2==0) || (((X1*Ec)**2+4*X2*Ec*(T-Hp))<0))
     Qc = X11*Ec/(T-Hp);          /* fall back 1st order mode */
else Qc= (2*X2*Ec)/(sqrt((X1*Ec)**2+4*X2*Ec*(T-Hp))-X1*Ec)
                          /* 2nd order mode */

Qc = Min (ceil(Ql*1.25), Qc, 31); /* clipping*/
Qc = Max (ceil(Ql*0.75), Qc, 1);  /* clipping */
Quantisation();
Encoding();
```

Step 4: After encoding the current frame

```
B += Rc - Rp;          /* update buffer fullness */
Rr -= Rc;              /* update the remaining bit counts */
S = Rc;            /* update the previous bit counts */
Hp = Hc;               /* update the previous header and motion bit counts */
Ql = Qc;               /* update the previous quantisation level */
Nr--;                  /* update the frame counter */
UpdateRDModel(Qc,Rc,Ec,Hc,X1,X2);
                  /* estimation of a new model */
while (B > 0.8 * Bs) {
     skip_next_frame();
     Nr--;
     B -= Rp;
}
```

If the buffer has reached 80% of the buffer size, the encoder will skip the upcoming frame to be encoded. Thus the buffer is reduced to prevent from the buffer overflow.

```
UpdateRDModel (Qc, Rc, Ec, Hc, X1, X2) {
     Qall[n]:  quantisation levels for the past frames
     Rall[n]:  scaled encoding complexity used for the past frames
     n: number of encoded past frames;
     x: matrix contains Q;
     y: matrix contains Q*(R-H)/E;
     Ep: mean absolute difference for the previous frame. This is computed after motion compensation for
     the Y component only.

     Rall[n] = (Rc-Hc)/Ec;
     Qall[n] = Qc;

     w = Min(total_data_number, 20) /* Maximum data number set to 20 */
     if(Ep>Ec)
        w = ceil(Ec/Ep*w);          /* sliding window for scene change */
     else
        w = ceil(Ep/Ec*w);          /* sliding window for scene change */
     Ep = Ec;                    /* update mad */

     for(i=n; i>n-w; i++){
          Qp[i]=Qall[i]
          Rp[i]=Rall[i]
     }
     Estimator(Qp,Rp,w,b);
     RemoveOutlier(X1,X2,Qp,Rp,Ec,w);
     Estimator(Qp,Rp,w,b);
}
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Estimator(Qp, Rp,w,b) {
    X11: First order coefficient.
    x = [1, Qp[i]**(-1)( i=1,2,...w)]          /* Dimension wx2 */
    y = [Qp[i]*Rp[i] (i=1,2,.....w)]       /* Dimension wx1 */

    for (i=1; i<=w; i++)   X11 += y[i]/w;

    if (all Qp[i] are not all the same) {
        b = (x_Transpose*x)**(-1)*x_Transpose*y;
        /* Dimension of the matrix */
        /* 2x1 = (2xw * wx2)**(-1)*(2xw)*(w*1) */
        X1 = b(1,1);
        X2 = b(2,1);
    }
}

RemoveOutlier(X1,X2,Qp,Rp,Ec,w) {
    error[w];      Estimation error
    new_w;

    for (i=1; i<=w; i++) {
        std += ((X1*Ec*Qp[i]**(-1)+X2*Ec*Qp[i]**(-2)-Rp[i]*Ec))**2;
        error[i] = X1*Ec*Qp[i]**(-1)+X2*Ec*Qp[i]**(-2)-Rp[i]*Ec;
    }

    /* When w is 2 the threshold should be 0, some implementation may resolve a small number due to
    precision problem. */
    threshold = sqrt(std/w);              /* Setup rejection threshold */

    new_w = 0
    for (i=1; i<=w; i++){
        if(abs(error[i]) <= threshold){
            new_w++;
            Qp[new_w] = Qp[i];
            Rp[new_w] = Rp[i];
        }
    }
    w = new_w;
}
```

L.1.3 Summary

The scalable rate control scheme achieves frame level rate control for both VBR and CBR cases. It assumes a simple quadratic rate distortion function of the video encoder. In the case of CBR encoding, a variable frame rate approach is used to achieve the target rate. If a tighter rate control is desired, the same technique is applicable at either slice layer or macroblock layer. Because of the generality of this scheme, extension of such a scheme to shape coding and Multiple-VO coding can be easily made on the basis of the existing rate control framework.

L.2 Multiple Video Object Rate Control

This clause describes an algorithm which achieves a constant bit rate when coding multiple video objects. The scheme assumes the following relationship between the bits and quantiser scale for the i-th object:

```
R[i]=X1[i]*S[i]*Q[i]**(-1)+X2[i]*S[i]*Q[i]**(-2)
```

The above is a simple extension of the relationship used for S-VOP rate control when sprite_enable == ‘static’.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

As shown in Figure L.1, the algorithm for performing the joint rate control can be decomposed into a number of steps, which may be grouped into a pre-encoding and post-encoding stage. Each block is briefly described below.

L.2.1 Initialization

The initialization process is very similar to the S-VOP process when sprite_enable == 'static'. Since a single buffer is used, the buffer drain rate is the same. The major difference is that most parameters are extended to vector quantities.

L.2.2 Quantisation Level Calculation for I-frame and first P-frame

To calculate the quantiser scales which are used for the I-frame and first P-frame, a bit allocation table based on Human Visual Sensitivity (HVS) of color tolerance is used. This scheme assigns a target bit rate which is predicted by histogram of variance and HVS classification.

Color/Variance classification: A macro block is classified into one of 32 macro block classes based on HVS and one of 16 classes based on the variance. A reference quantisation step size is assigned to the macro block according to its macro block class. A macro block is classified into a sensitive category if the number of pixels in the macro block which satisfy LUT (Look-Up Table) is larger than a threshold value. Otherwise the macro block is classified into insensitive category.

$$texture_class = H_n + V_n$$

$$H_n = \sum_{y=1}^{16}\sum_{x=1}^{16}\left(\left|P_{x,y} - P_{x-1,y}\right| > th\right)$$

$$V_n = \sum_{y=1}^{16}\sum_{x=1}^{16}\left(\left|P_{x,y} - P_{x,y-1}\right| > th\right)$$

$$th = f\left(\sum_{y=1}^{16}\sum_{x=1}^{16} P_{x,y}\right)$$

$$color_class = \min\left(color_tolerance\right)$$

$$color_tolerance = LUT\left(color_{mean}\right)$$

$$color_{mean} = \frac{1}{64}\sum_{y=1}^{16}\sum_{x=1}^{16}\left(P_{x,y}\right)$$

The block variance classification allows us to estimate the number of coded bits in a block. Generally, the number of coded bits generated in a block is proportional to the block variance when the quantisation step size is assumed to be constant. We divide the magnitude of block variance into 16 intervals. Threshold values in each interval can be found in Table L.1. Table L.2 illustrates the reference quantisation step size based on HVS for each macro block class and picture type.

Table L.1 — Block variance classes

Bv VOP	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
I	16	32	64	128	256	384	512	768	1024	1280	1792	2048	3072	4096	6154	¥
P	8	16	32	48	64	96	128	160	192	256	320	448	640	896	1536	¥
Bv : Block Variance Class Number, I : INTRA FRAME TYPE, P : INTER FRAME TYPE																

Table L.2 — Reference quantisation step size

Mc	0	1	2	3	4	5	6	7
Qss	-4	-2	-1	0	1	2	3	4

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Initial QP setting method

```
SetInitialQP()
{
    PreDiff = 1000000;
    Classification();
    if(Flat Image)        TargetBit = AverageBitPerFrame * 9;
    else                  TargetBit = AverageBitPerFrame * 4;

    /* Find Optimal Qp value */
    for(i=1; i<31; i++){
      SetMB_QP(i);  /* Allocate HVS optimal QP to each MB */
      EstimatedBit = Bitestimation(); /* Pre-bit-estimation */
      Diff = abs(TargetBit - EstimatedBit);
      if(Diff<PreDiff){
         PreDiff = Diff;
         MinErrorQ = i;
      }
    } /*  MinErrorQ is the final average Qp value */
    SetMB_QP(MinErrorQ); /* Allocate HVS Optimal Qp to each MB */
}
```

Frame target bit allocation

$$\text{FrameTargetBit} = \begin{Bmatrix} \frac{bitrate}{fps} * (4 \sim 9) ----\text{Intra frame} \\ \frac{bitrate}{fps} -----\text{First Inter Fr.} \end{Bmatrix}$$

Frame coded-bit estimation: There are 512 classes which are combinations of 32 macro block color classes(Mc) and 16 block variance classes(Bv). We can estimate the total number of coded bits in a frame with the histogram and the bit model table. **FrameEstimatedBit** is the estimated number of coded bit in the current frame

$$\text{FrameEstimateBit} = \sum_{Mc=0}^{31} \sum_{Bv=0}^{15} \{if\,(Coded) B[R[Mc]][Bv] * H[Mc][Bv]\} + \text{ShapeBit}$$

In the above equation, B[Mc][Bv] is the number of bits in the bit table and H[Mc][Bv] is the computation result of the histogram when a block is classified into Bv. The histogram can be computed with the information of macro block classes and block classes of the previous frame because the current frame to be coded and the previous frame are very highly correlated. Therefore the real-time coding process is possible.

Bit Table

The number of macro block classes and block classes in a frame depends on the statistical characteristics of the frame. However, the target number of bits which is assigned to the frame by bit allocation is not related with statistical characteristics of the frame. Hence the target number of bits for frame has much difference from the total number of coded bits in the frame when all macro blocks in the frame are quantised with their reference quantisation step sizes. So reference quantisation step sizes should be updated in order that the total number of coded bit in the frame becomes almost the same as the number of target bits for the frame. To determine the ratio of change of reference quantisation step size, we have to estimate the total number of coded bits in a frame when all macro blocks in the frame are quantised with their reference quantisation step sizes. The bit table shows the average value of the estimated number of bits in a block according to macro block classification and block classification in Intra/Inter frame. It is possible to estimate the total number of coded bits in a frame with the bit table.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
(INTRA FRAME Bit Table)
int BitTableI[32][16] =
{
65,145,174,208,242,278,298,317,341,352,365,386,400,419,435,438,
24, 79,102,127,153,179,194,209,232,243,255,266,272,279,286,292,
16, 52, 70, 93,114,136,148,161,180,190,201,214,220,226,231,234,
12, 38, 53, 73, 91,110,122,132,150,159,166,179,185,188,193,195,
11, 29, 42, 59, 75, 92,104,112,131,138,144,154,158,160,163,168,
10, 24, 35, 49, 64, 79, 91, 98,116,122,127,135,137,142,148,159,
 9, 19, 29, 42, 55, 69, 80, 87,103,109,116,122,126,130,137,147,
 9, 17, 25, 37, 48, 61, 72, 79, 95,100,105,111,114,119,125,135,
 9, 15, 22, 33, 43, 55, 65, 71, 86, 91, 97,103,106,110,115,123,
 8, 13, 19, 29, 39, 50, 59, 65, 79, 84, 89, 95, 98,100,103,114,
 8, 12, 18, 26, 35, 45, 54, 60, 73, 78, 84, 88, 89, 95, 99,104,
 8, 11, 16, 24, 32, 42, 51, 55, 68, 74, 79, 83, 85, 91, 94, 98,
 8, 11, 15, 22, 30, 38, 47, 51, 63, 69, 74, 78, 81, 86, 88, 90,
 8, 10, 14, 21, 28, 35, 44, 48, 59, 64, 69, 75, 77, 79, 80, 83,
 8, 10, 13, 19, 26, 33, 41, 45, 55, 61, 65, 71, 71, 74, 76, 78,
 8,  9, 12, 18, 24, 31, 39, 42, 52, 58, 61, 66, 66, 70, 72, 75,
 8,  9, 12, 17, 23, 29, 37, 40, 48, 55, 57, 62, 63, 65, 69, 73,
 8,  9, 11, 16, 21, 27, 35, 37, 46, 51, 55, 60, 60, 64, 66, 69,
 8,  9, 11, 14, 20, 26, 33, 36, 43, 48, 52, 56, 59, 63, 65, 66,
 8,  9, 10, 14, 19, 25, 31, 34, 41, 45, 50, 55, 57, 61, 64, 66,
 8,  9, 10, 13, 18, 23, 30, 33, 40, 44, 48, 53, 54, 59, 62, 63,
 8,  8, 10, 12, 17, 22, 28, 31, 38, 42, 46, 50, 52, 57, 60, 62,
 8,  8,  9, 12, 16, 21, 27, 30, 36, 40, 44, 49, 51, 55, 57, 60,
 8,  8,  9, 11, 16, 21, 26, 29, 35, 39, 43, 46, 48, 54, 55, 58,
 8,  8,  9, 11, 15, 20, 25, 28, 33, 38, 41, 44, 45, 51, 54, 57,
 8,  8,  9, 11, 14, 19, 24, 27, 32, 36, 40, 43, 44, 48, 50, 55,
 8,  8,  9, 10, 13, 18, 24, 26, 31, 35, 38, 42, 43, 46, 48, 55,
 8,  8,  8, 10, 13, 17, 23, 25, 29, 33, 37, 41, 41, 43, 46, 55,
 8,  8,  8, 10, 13, 17, 22, 24, 29, 32, 36, 39, 40, 42, 41, 53,
 8,  8,  8,  9, 12, 16, 22, 23, 28, 31, 35, 38, 39, 40, 43, 52,
 8,  8,  8,  9, 12, 16, 21, 22, 27, 30, 34, 36, 37, 40, 43, 52,
};
```

L.2.3 Update Rate-Distortion Model

After the encoding stage, the parameters for the RD model are updated. Given the number of texture bits, R[i], the quantisation level, Q[i], and the mean absolute difference, S[i], for the current and a specified number of past frames, the model parameters, X1[i] and X2[i] can be calculated using a least-squares estimation.

L.2.4 Post-Frameskip Control

The post-frameskip control determines the value of *N_skip_post.* It is invoked after the buffer has been updated, i.e., the sum of bits used to code every VO is added to the current buffer level and the number of bits to be removed per picture is subtracted. Essentially, if the “virtual buffer” has exceeded a specified level (e.g., 80% of the buffer size), the encoder will skip the upcoming frames so as to reduce the current buffer level. The virtual buffer level (VB) is given by:

$$VB = B + Bp - (N_skip_post+1)*Rp$$

where the new variable, Bp, denotes the number of bits used in the previous frame. As before, B is the current buffer level and Rp is the number of bits removed from the buffer per picture. After the value of *N_skip_post* has been found, the value of *N_skip_pre* is added to it, where *N_skip_pre* is calculated in the pre-frameskip control. The time instant is updated based on the total frames to be skipped, *N_skip* = *N_skip_pre* + *N_skip_post*.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Mode of Operation: Since coding decisions may change according to the environment, it is desirable to have a mechanism to detect and keep an update of such changes.

```
if(N_skip > SkipTH)
    Operate in LowMode
else
    Operate in HighMode
```

The mode of operation can be used to impose constraints on the coding parameters, assist in setting the weights for target distribution, and influence the shape rate control mechanism.

Target Estimation: The initial target bit rate can be calculated based on the number of available bits for the segment and the number of bits used in the previous corresponding VO. A similar lower bound to the S-VOP simulation can be used to target minimum quality.

Joint Buffer Control: The joint buffer control symbolizes the same operation as buffer control in S-VOP with sprite_enable == 'static', however, the target is a sum of individual targets, i.e., T = sum of T[i].

Pre-FrameSkip Control: Often, in low bit rate coding conditions, the target which emerges from the joint buffer control may not be enough to even code the motion, shape, and header information, let alone the texture. In that case, there should be a mechanism to alert other parts of the system (e.g., target distribution, QP calculation, time instant update) that there is a deficiency in the number of allocated bits. Let s = T - Hp be the difference between the target and the amount of bits used in the previous frame for shape, motion, and header, and Beta be a bit threshold greater than zero. N_skip_pre is determined by:

```
while(s < Beta)
    N_skip_pre = N_skip_pre + 1
    s = s + Rd
```

Target Distribution: The joint buffer control provides a new total target to be distributed among each VO. The distribution can be based on the size, the motion, and the variance (MAD^2) of each object:

$$T[i] = (MOTION[i]*wm + SIZE[i]*ws + MAD2[i]*wv)*T$$

This type of distribution is only executed when the target is valid. The target is valid when it is greater than the sum of header bits used for coding the previous frame. If it is not valid, all the targets are made negative. The negative targets can serve as a flag for clipping the quantiser scales to lie in a specified range [Qc 31].

Shape Rate Control: The value of AlphaTH has considerable effect over the number of bits which will be spent on the shape information. Using the current mode of operation, an appropriate value can be determined.

- If the mode of operation is LowMode, then increment the current AlphaTH by AlphaINC;
- If the mode of operation is HighMode, then decrement the current AlphaTH by AlphaDEC.

The minimum value of AlphaTH is 0. This implies lossless shape encoding. The largest value that AlphaTH may take is 255, however, it is recommended that an AlphaMAX in the range [32, 64] be specified.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

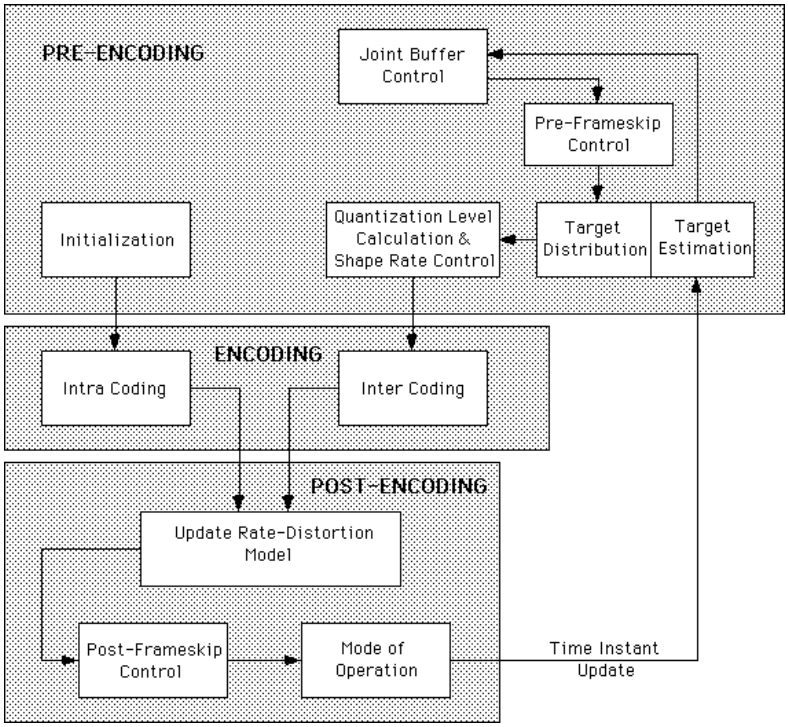


Figure L.1 -- Block diagram for multi-VO rate control

L.3 Macroblock Rate Control

The rate control method described in Clause L.1 used a fixed quantiser scale QP for all the macroblocks in a VOP, where the value of QP was chosen to encode the VOP with a target number of bits. In this clause we describe a method for adapting the value of QP at each macroblock within the VOP. This technique achieves the VOP target more accurately, which is useful for low-delay video applications with small buffer constraints, and allows adapting QP with the macroblock energy and other perceptually relevant measures. The method attempts to combine the strengths of the following contributions:

- A model of the human visual system as described in Clause L.2 can be used to compute W_j.
- The model with two parameters maintains a near-constant value of QP for all the macroblocks in the VOP. This approach is more appropriate at lower bit rates, since it requires a small number of bits for encoding the QP's (using DQUANT).
- The model with single parameter adapts QP with the energy of the block by using finer quantisation (smaller QP) for macroblocks of flatter image regions. It requires a larger QP bit overhead and hence is more appropriate at higher bit rates.

L.3.1 Rate-Distortion Model

We model the number of bits B_i produced for the i-th macroblock as follows:

$$B_i = \begin{cases} \frac{A_1}{QP_i^2} MAD_i^2; & \text{Bitrate} > R \\ \frac{A_2}{QP_i^2} MAD_i + \frac{A_3}{QP_i} MAD_i; & \text{Bitrate} \le R \end{cases} \quad (1)$$

- A1 , A2 and A3 are the model parameters.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- Bitrate is the rate available (in bits per pixel) for the texture of the VOP.
- R is a threshold. By default, we set R=0.085.

L.3.2 Target Number of Bits for Each Macroblock

Let N be the number of macroblocks in a VOP. The target number of bits, T_i, for the i-th macroblock is:

$$T_i = \frac{W_i \, MAD_i}{\sum_{j=1}^{N} W_j \, MAD_j} T_{TEX}, \qquad (2)$$

where:

- W_i is a weight that indicates the perceptual importance of the i-th macroblock. By default, we set $W_1 = W_2 = \ldots = W_N = 1$, but they could be obtained from similar techniques as described in Clause L.2.
- T_{TEX} is the target number of bits for the texture or DCT coefficients (luminance and chrominance components) of the VOP, which is the target frame bits T from frame-layer rate control [see Clause L.1] minus the bits for motion, shape, and syntax overhead. If the latter number of bits is not available for the current VOP, one can subtract those from the previous VOP. Bitrate in (1) is T_{TEX} divided by the number of pixels in the VOP.

Observe that if we combine (1) and (2), we obtain the following expression:

$$QP_i^2 = \begin{cases} MAD_i \, C_1; & Bitrate > R \\ C_2; & Bitrate \le R \end{cases} \qquad (3)$$

where the values of C_1 and C_2 depend on { A_1, MAD_1, …, MAD_N, T } and {A_2, A_3, MAD_1, …, MAD_N, T}, respectively. The model parameters A_1, A_2, and A_3 are estimated while encoding and in practice they are nearly constant within a VOP, and hence C_1 and C_2 are also approximately constant. As a result, observe that for large Bitrate, the quantiser scale QP will increase with the MAD energy of the macroblock, and for low Bitrate, QP will be nearly constant for all the macroblocks in the VOP.

L.3.3 Macroblock Rate Control

Step 1 Initialization.

N - number of macroblocks in the VOP,
T_{TEX} - number of bits available for encoding the VOP texture

If first frame, let A1= A1_prev= 100, A2= A2_prev= 400, A3= A3_prev= 0
Otherwise, let A1= A1_prev , A2= A2_prev, and A3= A3_prev.

If Bitrate > R = 0.085, let K=1, otherwise let K=2.
Set macroblock index i=1 and skip= 0.
Finally, let $S_1 = \sum_{j=1}^{N} W_j \, MAD_j$. (By default, set W1= W2= …=WN = 1.)

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Step 2 Compute QP for i_th macroblock

Let $T_i = \frac{W_i\, MAD_i}{S_i} T_{TEX}$, the target number of bits per macroblock and:

$$QP = \begin{cases} MAD_i \sqrt{\frac{A_1}{T_i}}; & K = 1 \\ Q: \frac{A_2}{Q^2} MAD_i + \frac{A_3}{Q} MAD_i = T_i; & K = 2 \end{cases}$$

If $(A_3 MAD_i)^2 + 4\, T_i A_2 MAD_i$ is negative, there is no value of Q that solves the second-order equation for K=2. In that case, the term with A_2 is dropped and the following is used instead:

$$QP = \frac{MAD_i A_{3_X11}}{T_i},$$

where A_{3_X11} is similar to the X11 parameter in the frame-based rate control in subclause L.1.2.

Round QP to value in 1, …, 31, and indicate the difference between the current value of QP and that of the previous macroblock in DQUANT. (Recall that DQUANT can only take values within {-2, -1, 0, 1, 2 }, so clipping may be needed.)

Step 3 Encode Macroblock with QP

Step 4 Update Counters

Let B_i' be the actual number of texture (DCT) bits produced when encoding the i_th macroblock. Compute:

$$T_{TEX} = T_{TEX} - B_i', \text{ and } S_{i+1} = S_i - W_i\, MAD_i$$

Step 5 Update A_1, or A_2, A_3, A_{3_X11}

If B_i' is zero (the DCT produced no bits), then skip= skip +1, and go to Step 6.

Otherwise, do the updating:

If K = 1, $\hat{A}_1 = \frac{B_i'}{MAD_i^2} QP^2$.

Let $A_1' = \hat{A}_1 \frac{1}{i - skip} + A_1' \frac{i - skip - 1}{i}$, $A_1 = A_1' \frac{i}{N} + A_{1_prev} \frac{N - i}{N}$.

If K = 2, use linear regression to estimate A2, A3, and A3_X11, similarly as in subclause L.1.2:
We order the values of B'i, QP, and MADi, for the last W encoded macroblocks (whose B'i > 0) into three sets: { b1, b2, b3, …, bW }, { q1, q2, …, qW }, and { m1, m2, …, mW }, where the bj's, qj's and mj's correspond to the values of B'i, QP, and MADi, respectively, for the j-th macroblock in the set.

Let NC be the number of macroblocks (whose B'i > 0) and set W= min{20, NC }. Define:

P1= [1, 1, …, 1]T,
P2= [1/q1, 1/q2, …, 1/qW]T,
P3= [b1q1/m1, b2q2/m2, …, bWqW/mW]T,
P4= [P1 P2],

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

where the superscript “T” denotes the transpose of a vector or matrix. P1, P2 and P3 are vectors of dimension W x 1, and P4 is a matrix of W x 2 elements.

Compute:

$$\begin{bmatrix} A_2 \\ A_3 \end{bmatrix} = \left(P4^T \cdot P4\right)^{-1} P4^T \cdot P3 ,$$

and $$A_{3_X11} = \frac{P3^T \cdot P2}{W} ,$$

where the superscript “-1” denotes the inverse of the matrix.

Step 6. If i = N, Stop (all macroblocks are encoded), let A_f_prev = A_f for f =1,2,3.

Otherwise, i = i+1, and go to Step 2.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex M
(informative)

Binary shape coding

M.1 Introduction

This part of ISO/IEC 14496 will be the first international standard allowing the transmission of arbitrarily shaped video objects (VO). A time instant of a VO is called Video Object Plane (VOP). A VOP is a rectangular video frame or a part thereof. Following an object-based approach, the ISO/IEC 14496-2 video encoder applies the motion, texture and shape coding tools to the VOP using I, P, and B modes similar to the modes of MPEG-2. The encoder transmits texture, motion, and shape information of one VO within one bit stream. The bit streams of several VOs and accompanying composition information can be multiplexed such that the decoder receives all the information to decode the VOs and arrange them into a video scene (Figure M.1).

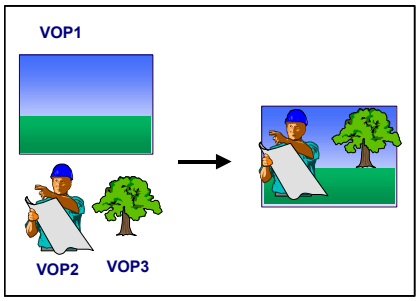


Figure M.1 — Object-based coding requires the decoder to compose different Video Object Planes (VOP) into a scene.

An arbitrarily shaped VO can have binary shape information only or binary shape and texture information. Several shape coding algorithms were evaluated [1][2]. Binary shape information is transmitted as a bitmap using a context-based arithmetic coder described in Clause M.2. Coding of texture for arbitrarily shaped VOs is described in Clause M.3. This part of ISO/IEC 14496 leaves it up mainly to the encoder to achieve efficient coding of texture at object boundaries by employing an appropriate texture extrapolation algorithm. The efficient coding of a VO with lossily encoded shapes requires an encoder architecture as described in Clause M.4. In Clause M.5, guidelines for coding arbitrarily shaped VOs will be presented. Different shape encoding modes are investigated in order to achieve a bit efficient representation of a VO or in order to allow for fast encoding that is important for real-time applications.

M.2 Context-Based Arithmetic Shape Coding

In order to enable content based access to video objects, this part of ISO/IEC 14496 codes the shape of video objects. This part of ISO/IEC 14496 chose to code the shape as a bitmap [2]. For binary shape coding, a rectangular bounding rectangle enclosing the arbitrarily shaped VOP is formed such that its horizontal and vertical dimensions are multiples of 16 pels (macroblock size).

Each block of size 16x16 pels within this bounding rectangle is called binary alpha block (bab). Each bab is associated with the co-located macroblock. Three types of babs are distinguished and signaled to the decoder: Transparent blocks do not contain information about the object, opaque blocks are located entirely inside the object and boundary blocks cover part of the object as well as part of the background. For boundary blocks a context-based shape coder was developed. This coder exploits the spatial redundancy of the binary shape information to be coded. Pels are coded in scan-line order and row by row. In the following paragraphs, shape encoding in intra mode is described first. Then, this technique is extended to include an inter mode.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M.2.1 Intra Mode

In intra mode, three different types of macroblocks are distinguished: Transparent and opaque blocks are signaled as macroblock type. The macroblocks on the object boundary containing transparent as well as opaque pels belong to the third type. For these boundary macroblocks, a template of 10 pels is used to define the causal context for predicting the shape value of the current pel (Figure M.2a). For encoding the state transition, a context-based arithmetic encoder corresponding to the arithmetic decoder as defined in subclause 7.5.3 is used. The probability table of the arithmetic encoder for the 1024 contexts was derived from sequences that are outside of the test set used for comparing different shape coders. With two bytes allocated to describe the symbol probability for each context, the table size is 2048 bytes. In order to avoid emulation of start codes like VOP start code, the arithmetic coder stuffs one '1' into the bitstream whenever a long sequence of '0' is sent.

The template extends up to 2 pels to the left, to the right and to the top of the pel to be coded (Figure M.2a). Hence, for encoding the pels in the 2 top and left rows of a bab, parts of the template are defined by the shape information of the already transmitted babs on the top and on the left side of the current bab. For the 2 right-most columns, each undefined pel of the context is set to the value of its closest neighbor inside the bab. In case the top and left babs are transparent, the top and left 2 rows of the current bab are duplicated in order to define the context for the template for intra mode.

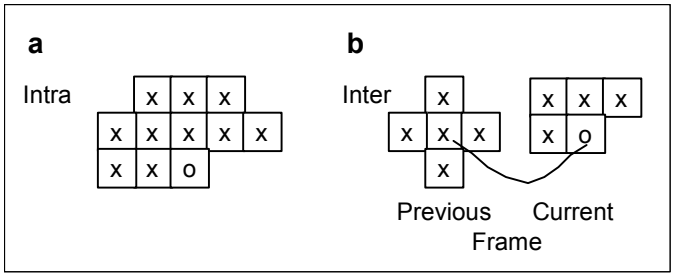


Figure M.2 — Templates for defining the context of the pel to be coded (o),
a) defines the intra mode context,
b) the context when coding in inter mode.
The alignment is done after motion compensating the previous VOP [1].

In order to increase coding efficiency as well as to allow lossy shape coding, a bab can be subsampled by a factor of 2 or 4 resulting in a sub-block of size 8*8 pels or 4*4 pels, respectively. The sub-block is encoded using the encoder as described above. The encoder transmits to the decoder the subsampling factor such that the decoder decodes the shape data and then upsamples the decoded sub-block to macroblock size. Obviously, encoding the shape using a high subsampling factor is more efficient, but the decoded shape after upsampling may or may not be the same as the original shape. Hence, this subsampling is mostly used for lossy shape coding. In order to achieve smooth object boundaries after upsampling, an adaptive non-linear upsampling filter is used. The context of this upsampling filter is shown in Figure M.3.

The efficiency of the shape coder differs depending on the orientation of the shape data. Therefore the encoder can choose to code the bab using horizontal scanning as described above or transpose the macroblock using vertical scanning prior to arithmetic coding.

M.2.2 Inter Mode

In order to exploit temporal redundancy in the shape information, the coder described above is extended by an inter mode requiring motion compensation and a different template for defining the context.

For motion compensation, a 2D integer pel motion vector is estimated using full search for each bab in order to minimize the prediction error between the previous coded VOP shape M'k-1 and the current shape Mk. The shape motion vectors are predictively encoded with respect to the shape motion vectors of neighboring macroblocks. If no shape motion vector is available, texture motion vectors are used as predictors. The shape motion vector of the current block is used to align a new template designed for coding shape in inter mode (Figure M.2b). The template defines a context of 9 pels resulting in 512 contexts. The probability for one symbol is described by 2 bytes giving a probability table size of 1024 bytes. Four pels of the context are neighbors of the pel to be coded, 5 pels are located at the motion compensated location in the previous VOP. Assuming that the motion vector (dx,dy)T points

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

from the current vopk to the previous coded vop'k-1, the part of the template located in the previously coded shape is centered at m'(x-dx,y-dy) with (x,y)T being the location of the current pel to be coded.

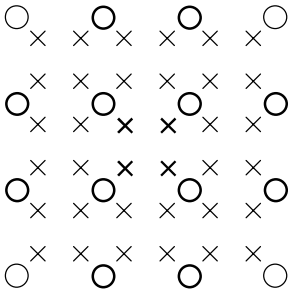


Figure M.3 — For shape upsampling, the upsampled pels (x) lie between the location of the subsampled pels (o). Neighboring pels (bold o) defining the values (transparent or opaque) of the pels to be upsampled (bold x).

In inter mode, the same options as in intra mode like subsampling and transposing are available. For lossy shape coding, the encoder may also decide that the shape representation achieved by just carrying out motion compensation is sufficient thus saving bits by avoiding the coding of the prediction error. The encoder can select one of 7 modes for the shape information of each bab: transparent, opaque, intra, inter with and without shape motion vectors, and inter with/without shape motion vectors and prediction error coding. These different options with optional subsampling and transposition allow for encoder implementations of different coding efficiency and implementation complexity.

M.3 Texture Coding of Boundary Blocks

For motion compensated prediction of the texture of the current VOP, the reference VOP is motion compensated using block motion compensation. In order to guarantee that every pel of the current VOP has a value to be predicted from, all of the boundary blocks and some of the transparent blocks of the reference VOP have to be padded using motion compensation (MC) padding. Boundary blocks are padded using repetitive padding: First boundary pels are replicated in horizontal direction, then in vertical direction making sure that if a value can be assigned to a pel by both padding directions an average value is assigned to the pel. Since this repetitive padding puts a significant computational burden on the decoder, a simpler mean padding is used in a second step. Transparent macroblocks bordering boundary blocks are assigned to an average value determined by the pels of its neighboring padded blocks.

In order to encode the texture of a boundary block, this part of ISO/IEC 14496 treats the macroblock as a regular macroblock and encodes each block using an 8*8 DCT. The decoder decodes the texture and discards all information that falls outside of the decoded shape. In order to increase coding efficiency, the encoder can choose the texture of pels outside of the object such that the bitrate is minimized. This non-normative process is called texture padding. For intra mode, a lowpass extrapolation filter increasing the PSNR of boundary blocks by an average of 2dB was developed [4]. For inter mode setting these pels to 0 gives good efficiency.

VOP texture may be encoded using B-frames. The related shape information of the B-frame is coded using the inter mode as described in below using the closest P or I frame as a reference VOP.

M.4 Encoder Architecture

Figure M.4a shows the block diagram of this object-based video coder. This diagram focuses on the object-based mode in order to allow a better understanding of how shape coding influences the encoder and decoder. Image analysis creates the bounding rectangle for the current VOP S_k and estimates texture and shape motion of the current VOP S_k with respect to the reference VOP S'_{k-1}. Parameter coding encodes the parameters predictively. The parameters get transmitted, decoded and the new reference VOP is stored in the VOP memory and also handed to the compositor of the decoder for display. The increased complexity due to the coding of arbitrarily shaped video objects becomes evident in Figure M.4b that shows a detailed view of the parameter coding.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Since this part of ISO/IEC 14496 defines motion vectors for motion compensation of texture and for motion compensation of shape, we refer to them as texture motion vectors and shape motion vectors, respectively. The parameter coder encodes first the shape of each boundary block using shape motion vectors and texture motion vectors for prediction. Lossy shape coding may change a boundary block to a transparent block. If the boundary block is not transparent, its shape motion vector is coded. The shape motion coder knows which motion vectors to code by analyzing the possibly lossily encoded shape parameters.

For texture prediction, the reference VOP is padded as described above. The prediction error is then padded using the original shape parameters to determine the area to be padded. Using the original shape as a reference for padding is again an encoder choice. Using the original or the coded shape for defining the texture of a macroblock makes obviously only a difference if the shape is coded lossily. Assuming that the texture between object and background in the original VOP differs significantly, using the original shape to define the area to be padded in the macroblock prevents the background texture from influencing the padding process. Texture padding will use the object texture for padding resulting in an efficiently to code block. If on the other hand the coded shape is used to define the area to be padded, background texture may affect the padding result potentially increasing the bitrate for texture coding drastically. Finally, the texture of each macroblock is encoded using DCT.

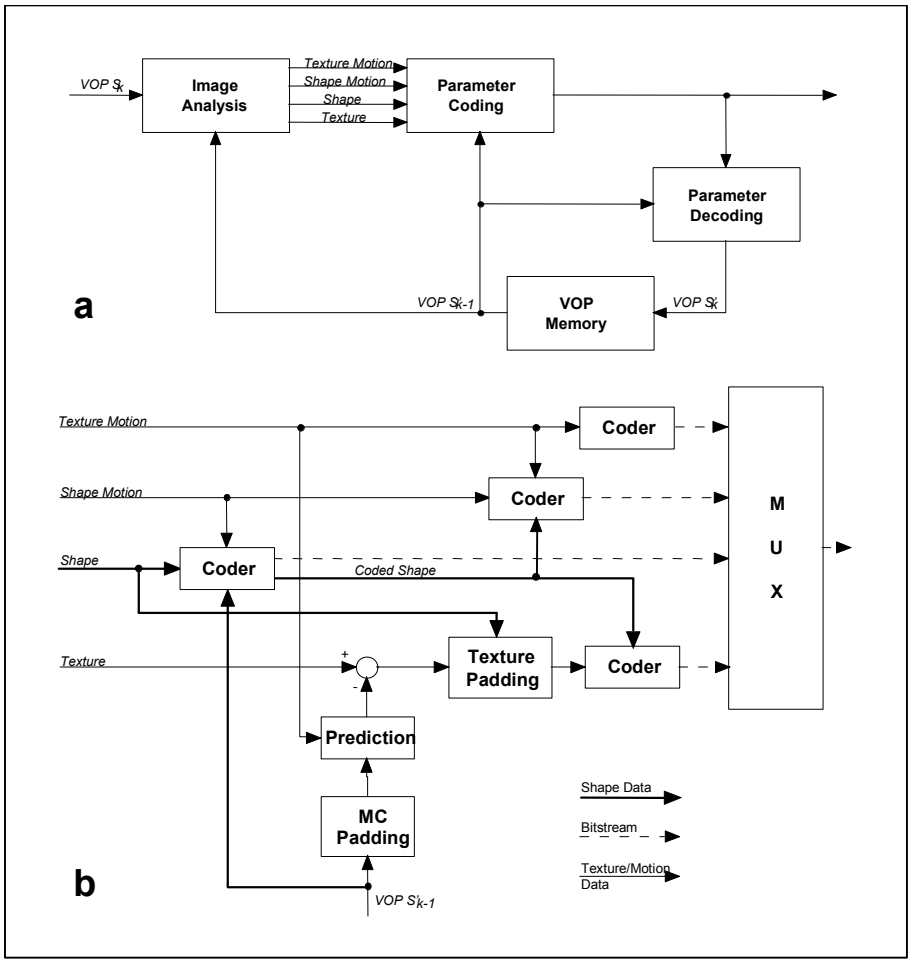


Figure M.4 — Block diagram of the video encoder (a) and the parameter coder (b) for coding of arbitrarily shaped video objects.

M.5 Encoding Guidelines

When encoding arbitrarily shaped VOs several new encoding options become available when compared to coding of rectangular video sequences. Here we provide some hints on lossy shape coding, shape coding mode selection.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

M.5.1 Lossy Shape Coding

The decoder has no possibility to find out whether the encoder uses lossless or lossy shape coding, what shape coding strategy the encoder uses or texture padding algorithm is used. Therefore, post-processing filter for the binary shape information may not be used. In its reference implementation of a video coder, this part of ISO/IEC 14496 chose to control lossy shape coding by using an alpha-threshold. The alpha threshold defines the maximum number of incorrectly coded pels within a boundary block. A fixed or adaptive alpha-threshold can be used for each bab. Using an alpha threshold may change the topology of the shape. Often, isolated pels at the object boundary are coded as part of the object. Using morphological filters at the encoder to smooth object boundaries or to filter the prediction error with respect to the original shape provides a much more predictable quality of a lossily encoded shape. Furthermore, the encoder can choose to neglect the prediction error signal at the object boundary if it does not change the topology of the object and to update the prediction error in those parts of the block where the prediction error leads to a change of the object topology.

M.5.2 Coding Mode Selection

As far as shape coding is concerned, this part of ISO/IEC 14496 allows for 18 coding modes of each BAB: (Intra/inter/inter with MC)*(horizontal/vertical scanning)*(Subsampling factor 0/1/2). The influence of different shape coding modes on the performance of the coder in terms of coding efficiency but also in terms of computational complexity is of interest, since the shape coder including padding can account for up to 50% of the encoder operations [3]. When encoding a binary shape, checking every mode achieves the highest compression. However, the gain achieved by this checking should be evaluated in relation to the overall bitrate required for coding a VOP. Table M.1 shows the main usage for each coding mode.

Table M.1 — Shape coding modes and their main usage

	Mode	Main Usage
1	intra	I frames, arbitrarily shaped still texture object, error resilience
2	inter, inter MC	P frames
3	horizontal/vertical scanning	Low-bitrate shape coding
4	subsampling to a block size 8x8 or 4x4	Low-bitrate lossy shape coding

The intra mode is used for I frames and for coding the shape of the arbitrarily shaped still texture object. Similarly, the use of the intra mode as well as the inter mode with and without motion compensation should be investigated for coding of P frames. For low-bitrate applications, the shape usually requires a significant percentage of the overall bitrate. Then it makes sense for the encoder to investigate the gain of adapting the scanning direction of each BAB reducing the shape bit rate by up to 8%. In case of lossy shape coding at low bit rates, the subsampling filter reduces the bit rate for shape coding by up to 20% for an alpha threshold of 16. The use of the subsampling filter is most effective for I babs. In case of lossless shape coding, the coder should signal at the beginning of each VOP that the subsampling feature would not be used. This avoids the overhead of signaling the subsampling for each BAB.

M.6 Conclusions

This part of ISO/IEC 14496 standardizes an object-based decoder. Several choices regarding texture extrapolation at object boundaries and shape encoding are available to the encoder. It was found that for texture extrapolation a simple filter setting the texture outside of the *original* object shape to gray is sufficient. In the case of lossy shape coding, not using the *original* object shape for defining the support region of the texture extrapolation may increase the overall bitrate by more than 100%. Therefore, lossy shape coding makes only sense if the encoder architecture presented in this paper is used.

Fixing the bab scanning to horizontal instead of adaptively switching between horizontal and vertical scanning increases the overall bitrate by just 1% but saves significant compute time since the compute intensive arithmetic coding with context determination has to be used only half as often.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The downsampling of the shape information increases the computational complexity of the encoder. In the case of lossy shape coding in intra mode, the subsampling increases the subjective quality of the coded object shape due to a non-linear shape upsampling filter. For low bitrate video coding using lossily coded shapes, this filter reduces the overall bitrate.

As the experiments show, the ISO/IEC 14496-2 binary shape coder can operate in a very bit efficient manner. In order to save computational complexity for high bit rate applications, the encoder may choose to not explore all of the encoding options and choose a less efficient shape encoding. The increase of the overall bitrate may be less than 1% at high bit rates but the computational complexity of the encoder decreases by 15% to 50%.

M.7 References

[1] N. BRADY, F. BOSSEN, N. MURPHY, "Context-based arithmetic encoding of 2D shape sequences", Special session on shape coding, ICIP 97, Santa Barbara, 1997

[2] A. KATSAGGELOS, L. KONDI, F. MEIER, J. OSTERMANN, G. SCHUSTER, "MPEG-4 and rate-distortion based shape coding techniques", Proceedings of the IEEE, pp. 1126-1154, June 1998

[3] J. OSTERMANN, "Coding of arbitrarily shaped video objects with binary and greyscale alpha maps: What can MPEG-4 do for you?". ISCAS 98, Monterey, CA, TPA 2-1, June 1998

[4] A. KAUP, "Adaptive low-pass extrapolation for object-based texture coding of moving video", Proc. Visual Communications and Image Processing, SPIE vol. 3024, pp. 731-741, Feb. 1997

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex N
(normative)

Visual profiles@levels

The table that describes the visual profiles and levels is given below, with the following notes:

1 Enhancement layers are not counted as separate objects.

2 The maximum VMV buffer size is the bound on the memory (in macroblock units) which can be used by the VMV algorithm. This algorithm (see Clause D.5) models the pixel memory needed by the entire visual decoding process. This includes memory needed for reference VOPs in the prediction of P, B, and S(GMC)-VOPs and the storage of the reconstructed VOPs until such time that they are released by the decoder, plus the memory required to queue B-VOPs until composition occurs. In profiles that contain more than one layer, the memory requirements include all base and enhancement layers. When belonging to different, overlapping objects, some of these macroblocks may overlay on the display, however separate memory is required (prior to composition) in the VMV.

3 The conformance point for the base layer of Simple Scalable Visual Profile is the Simple Profile @L1 when Simple Scalable Profile @L1 is used and the Simple Profile @ L2 when Simple Scalable Profile @ L2 is used.

4 The VCV decoder rate is the vcv_decoder_rate (H) referred to in Clause D.4; it is the number of macroblocks/second based on the typical spatial and temporal resolutions, as follows:
- 1485 MBs/s corresponds to QCIF at 15Hz.
- 5940 MBs/s corresponds to CIF at 15 Hz and also twice QCIF at 30 Hz.
- 11880 MB/s corresponds to CIF at 30 Hz.
- 7425 MB/s corresponds to 1.25 times CIF at 15 Hz.
- 23760 MB/s corresponds to twice CIF at 30 Hz.
- 97200 MB/s corresponds to twice ITU-R 601 at 30 Hz.
- 489600 MB/s corresponds to twice 1920x1088 at 30 Hz.

5 The total (aggregated) vbv_buffer_size is the sum of the individual VBV buffer occupancies at any given time (in units of 16384 bits) for all VOLs of all VOs. This total VBV size is limited according to the profile and level.

6 The maximum video packet length is defined as the maximum number of bits of data_partitioned_motion_shape_texture() in one video packet. The constraint applies only when the data-partitioning tool is enabled in the bitstream. When data partitioning is disabled, there is no limit on the size of video packet length.

7 N. A. means Not Applicable.

8 The maximum VCV buffer size (cumulative over all layers of all VOs) is twice the maximum number of macroblocks per VOP in the profile and level combination except for the Simple Visual Profile, Simple Scalable profile (Level 1) and Advanced Real Time Simple Profile. For the Simple Visual Profile and the Advanced Real Time Simple Profile, this value is maximum number of macroblocks per VOP. For the Simple Scalable profile (Level 1), it is 1.25 times of the maximum number of macroblocks per VOP. The limit applies to both VCV buffer and the boundary MB VCV buffer.

9 The VCV boundary MB decoder rate column bounds the number of macroblocks containing non trivial shape information. The VCV boundary MB decoder rate constrains the total number of boundary MBs in all VOLs concurrently. Note that the boundary macroblocks are added to both the VCV and boundary VCV buffers.

10 Max unique Quant Tables is for the total number of Quant Tables. It is the sum of intra, non-intra, grey-scale intra, and grey-scale non-intra Quant Tables used for all objects. The four types of tables can be on or off independently in the syntax definition.

--`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11 For Simple Profile @ Level L0, the following restrictions apply:

a) The maximum frame rate shall be 15 frames per second;

b) The maximum f_code shall be 1;

c) The intra_dc_vlc_threshold shall be 0;

d) The maximum horizontal luminance pixel resolution shall be 176 pels/line;

e) The maximum vertical luminance pixel resolution shall be 144 pels/VOP;

f) If AC prediction is used, the following restriction applies : QP value shall not be changed within a VOP (or within a video packet if video packets are used in a VOP). If AC prediction is not used, there are no restrictions to changing QP value.

12 For Simple Scalable Profile @ Level L0, the following restrictions apply:

a) Base layer compliant to Simple Profile @ Level L0;

b) The enhancement layer maximum f_code shall be 1;

c) The enhancement layer intra_dc_vlc_threshold shall be 0;

d) For enhancement layer, if AC prediction is used, the following restriction applies: QP value shall not be changed within a VOP (or within a video packet if video packets are used in a VOP). If AC prediction is not used, there are no restrictions to changing QP value;

e) Spatial scalability shall be limited to a 1:1 scaling ratio.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table N.1 — Definition of Version 1 Natural Visual Profiles@Levels

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[8]	VCV decoder rate (MB/s)[4]	VCV Bound-ary MB decoder rate (MB/s)[9]	Max total VBV buffer size (units of 16384 bits)[5]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
N-Bit	L2	CIF	16	16 x Core or Simple or N-Bit	4	2376	792	23760	11880	80	80	8192	N. A.[7]	N. A.	2000	1
Main	L4	1920 x 1088	32	32 x Main or Core or Simple	4	65344	16320	489600	244800	760	760	16384	65280	Scalable Texture Profile@L2	38400	1
Main	L3	ITU-R 601	32	32 x Main or Core or Simple	4	11304	3240	97200	48600	320	320	16384	6480	Scalable Texture Profile@L1	15000	1
Main	L2	CIF	16	16 x Main or Core or Simple	4	3960	1188	23760	11880	80	80	8192	1584	Scalable Texture Profile@L1	2000	1
Core	L2	CIF	16	16 x Core or Simple	4	2376	792	23760	11880	80	80	8192	N. A.	N. A.	2000	1
Core	L1	QCIF	4	4 x Core or Simple	4	594	198	5940	2970	16	16	4096	N. A.	N. A.	384	1
Simple Scalable[3]	L2	CIF	4	4 x Simple or Simple Scalable	1	3168	792	23760	N.A.	40	40	4096	N. A.	N. A.	256	1 spatial or temporal enhance ment layer
Simple Scalable	L1	CIF	4	4 x Simple or Simple Scalable	1	1782	495	7425	N. A.	40	40	2048	N. A.	N. A.	128	1 spatial or temporal enhance ment layer
Simple Scalable	L0	QCIF	1	1 x Simple or Simple Scalable	1	446	124	1865	N. A.	10	10	2048	N. A.	N. A.	128	1 spatial or temporal enhance ment layer

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[8]	VCV decoder rate (MB/s) [4]	VCV Bound-ary MB decoder rate (MB/s)[9]	Max total VBV buffer size (units of 16384 bits)[5]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
Simple	L3	CIF	4	4 x Simple	1	792	396	11880	N. A.	40	40	8192	N. A.	N. A.	384	N. A.
Simple	L2	CIF	4	4 x Simple	1	792	396	5940	N. A.	40	40	4096	N. A.	N. A.	128	N. A.
Simple	L1	QCIF	4	4 x Simple	1	198	99	1485	N.A.	10	10	2048	N. A.	N. A.	64	N. A.
Simple	L0	QCIF	1	1 x Simple	1	198	99	1485	N.A.	10	10	2048	N. A.	N. A.	64	N. A.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table N.2 — Definition of Version 2 Natural Visual Profiles@Levels

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[8]	VCV decoder rate (MB/s) [4]	VCV Bound-ary MB decoder rate (MB/s)[9]	Max total VBV buffer size (units of 16384 bits)[5]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
Advanced Coding Efficiency	L4	1920 x 1088	32	32 x Adv. Coding Efficiency or Core or Simple	4	48960	16320	489600	244800	760	760	16384	N. A.	N. A.	38400	1
Advanced Coding Efficiency	L3	ITU-R 601	32	32 x Adv. Coding Efficiency or Core or Simple	4	9720	3240	97200	48600	320	320	16384	N. A.	N. A.	15000	1
Advanced Coding Efficiency	L2	CIF	16	16 x Adv. Coding Efficiency or Core or Simple	4	2376	1188	23760	11880	80	80	8192	N. A.	N. A.	2000	1
Advanced Coding Efficiency	L1	CIF	4	4 x Adv. Coding Efficiency or Core or Simple	4	1188	792	11880	5940	40	40	8192	N. A.	N. A.	384	1
Core Scalable	L3	ITU-R 601	16	16 x Core or Simple or Core scalable or Simple Scalable	4	12906	4032	120960	60480	80	80	16384	N. A.	N. A.	4000	2
Core Scalable	L2	CIF	8	8 x Core or Simple or Core scalable or Simple	4	2970	990	29700	14850	80	80	4096	N.A.	N.A.	1500	1
Core Scalable	L1	CIF	4	4 x Core or Simple or Core scalable or Simple Scalable	4	2376	792	14850	7425	64	64	4096	N.A.	N.A.	768	1
Advanced Core	L2	CIF	16	16 x Core or Simple or Adv. scalable Texture	4	2376	792	23760	11880	80	40	8192	N. A.	see Table 9-10	2000	1
Advanced Core	L1	QCIF	4	4 x Core or Simple or Adv. Scalable Texture	4	594	198	5940	2970	16	8	4096	N. A.	see Table 9-10	384	1

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Visual Profile	Level	Typical Visual Session Size	Max ob-jects [1]	Maximum number per type	Max unique Quant Tables	Max. VMV buffer size (MB units)[2]	Max VCV buffer size (MB)[8]	VCV decoder rate (MB/s) [4]	VCV Bound-ary MB decoder rate (MB/s)[9]	Max total VBV buffer size (units of 16384 bits)[5]	Max vbv buffer size (units of 16384 bits)	Max. video packet length (bits)[6]	Max sprite size (MB units)	Wavelet restrictions	Max bitrate (kbit/s)	Max. enhance-ment layers per object
Advanced Real Time Simple	L4	CIF	16	16 x Simple or Adv. Real Time Simple	1	792	396	11880	N. A.	80	80	16384	N. A.	N. A.	2000	N. A.
Advanced Real Time Simple	L3	CIF	4	4 x Simple or Adv. Real Time Simple	1	792	396	11880	N. A.	40	40	16384	N. A.	N. A.	384	N. A.
Advanced Real Time Simple	L2	CIF	4	4 x Simple or Adv. Real Time Simple	1	792	396	5940	N. A.	40	40	16384	N. A.	N. A.	128	N. A.
Advanced Real Time Simple	L1	QCIF	4	4 x Simple or Adv. Real Time Simple	1	198	99	1485	N.A.	10	10	8192	N. A.	N. A.	64	N. A.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table N.3 — Definition of Studio Visual Profiles@Levels

Visual Profile	Level	Typical Visual Session formats[1]	Max. Pixel depth	Max. objects	Maximum number per type	Max. VMV buffer size (sample)[2]	Max. VCV buffer size (sample)[3]	VCV decoder rate (sample/s)	VCV Bound-ary MB decoder rate (sample/s)	Max. total VBV buffer size	Max. vbv buffer size	Max. video packet length (bits)	Max. sprite size (sample)[4]	Wavelet restric-tions	Max. bitrate (Mbit/s)	Max. enhance-ment layers per object
Simple Studio	L4	ITU‑R709.60P:444 ITU‑R709.60I:444444 2Kx2Kx30P:444	12	1	1 x Simple Studio	12582912	12582912	377487360	377487360	4320	4320	N.A.	N.A.	N.A.	1800	N.A.
Simple Studio	L3	ITU‑R709.60I:444 ITU‑R709.60I:4224	12	1	1 x Simple Studio	6291456	6291456	188,743680	188,743680	2880	2880	N.A.	N.A.	N.A.	900	N.A.
Simple Studio	L2	ITU‑R709.60I:422 ITU-R601:444444	10	1	1 x Simple Studio	4194304	4194304	125,829120	125,829120	1920	1920	N.A.	N.A.	N.A.	600	N.A.
Simple Studio	L1	ITU-R601:4224 ITU-R601:444	10	1	1 x Simple Studio	1313280	1313280	33177600	33177600	576	576	N.A.	N.A.	N.A.	180	N.A.
Core Studio	L4	ITU‑R709.60P:444 ITU‑R709.60I:444444 2Kx2Kx30P:444	10	16	16 x Core Studio or Simple Studio	50331648	25165824	754974720	754974720	4320	4320	N.A.	150994944	N.A.	900	N.A.
Core Studio	L3	ITU‑R709.60I:444 ITU‑R709.60I:4224	10	8	8 x Core Studio or Simple Studio	25165824	12582912	377487360	377487360	2880	2880	N.A.	75202560	N.A.	450	N.A.
Core Studio	L2	ITU‑R709.60I:422 ITU-R601:444444	10	4	4 x Core Studio or Simple Studio	16777216	8388608	251658240	251658240	1920	1920	N.A.	50135040	N.A.	300	N.A.
Core Studio	L1	ITU-R601:4224 ITU-R601:444	10	4	4 x Core Studio or Simple Studio	5253120	2626560	66355200	66355200	576	576	N.A.	8294400	N.A.	90	N.A.

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
--`,,`-`-`,,`,,`,`,,`---
Not for Resale
© ISO/IEC 2004 – All rights reserved

[1] ITU‑R709 is ITU-R BT. 709 and ITU‑R601 is ITU-R BT. 601
444444 means 444(RGB)+ 3 Auxiliary Channels
4224 means 422(YUV)+ 1 Auxiliary Channels

[2] VMV is defined by number of samples which belong to the bounding box of texture regardless shape information. VMV also includes auxiliary channel samples.

[3] VCV is defined by number of samples which belong to the bounding box of texture regardless shape information. VCV also includes auxiliary channel samples.

[4] Max sprite size is defined by number of samples for Sprite memory.

Within Advanced Simple Profile and Fine Granularity Scalable Profile, five levels are defined in each profile as specified in Table N.4.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table N.4 — Definition of Levels in Advanced Simple Profile and Fine Granularity Scalable Profile

Visual Profile	Level	Typical Visual Session Size	Max. objects	Maximum number per type	Max. unique Quant Tables	Max. VMV buffer size (MB units)	Max VCV buffer size (MB)	VCV decoder rate (MB/s)	Max. Percentage of Intra MBs with AC prediction in VCV buffer	Max total VBV buffer size (units of 16384 bits)	Maximum VBV Buffer Size (units of 16384 bits)	Max. video packet length (bits)	Maximum Bitrate (kbits/s) (Note 2)	Maximum number of coded vop-bps (Note 3)
AS	L0	176x144	1	1x AS or Simple	1	297	99	2970	100	10	10	2048	128	N.A.
AS	L1	176x144	4	4x AS or Simple	1	297	99	2970	100	10	10	2048	128	N.A.
AS	L2	352x288	4	4x AS or Simple	1	1188	396	5940	100	40	40	4096	384	N.A.
AS	L3	352x288	4	4x AS or Simple	1	1188	396	11880	100	40	40	4096	768	N.A.
AS	L3b	352x288	4	4x AS or Simple	1	1188	396	11880	100	65	65	4096	1500	N.A.
AS	L4	352x576	4	4x AS or Simple	1	2376	792	23760	50	80	80	8192	3000	N.A.
AS	L5	720x576	4	4x AS or Simple	1	4860	1620	48600	25	112	112	16384	8000	N.A.
FGS	L0	176x144	1	1x AS or FGS or Simple	1	297	99	2970	100	10	10	2048	128	4
FGS	L1	176x144	4	4x AS or FGS or Simple	1	297	99	2970	100	10	10	2048	128	4
FGS	L2	352x288	4	4x AS or Simple	1	1188	396	5940	100	40	40	4096	384	4
FGS	L3	352x288	4	4x AS or FGS or Simple	1	1188	396	11880	100	40	40	4096	768	4
FGS	L4	352x576	4	4x AS or FGS or Simple	1	2376	792	23760	50	80	80	8192	3000	4
FGS	L5	720x576	4	4x AS or FGS or Simple	1	4860	1620	48600	25	112	112	16384	8000	4

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Note1: The following restriction applies to L0 of AS Profile and FGS Profile:

- If AC prediction is used, QP value shall not be changed within a VOP (or within a video packet if video packets are used in a VOP). If AC prediction is not used, there are no restrictions to changing QP value.

Note 2: For FGS Profile, this column is the maximum base-layer bitrate.

Note 3: The maximum number of coded vop-bps takes into consideration the shifted bits after applying frequency weighting and/or selective enhancement.

Note 4: The number of FGS, FGST, or FGS-FGST layer is always one. If FGS layer and FGST layer are separated, the number of total enhancement layers is two.

Note 5: The interlace tools are not used for levels L0, L1, L2, and L3 of AS Profile and FGS Profile.

Note 6: It is inherent in the FGS profile that the base and enhancement layers are tightly coupled to one another. To avoid unnecessary memory storage, the following constraints apply to the decoding time relationship of the enhancement layer and the base layer:

- Decoding and composition (or presentation in a no-compositor decoder) of each FGS or FGST VOP shall be performed in the same time unit.
- Decoding of each FGS and FGST VOP shall be performed immediately after the reference base layer VOP(s) are decoded without violating the above constraint.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex O
(informative)

3D Mesh Coding

O.1 Introduction

Representation of a virtual world (including 3-D models) is not something new in this part of ISO/IEC 14496. For example, ISO/IEC 14772-1:1997 describes an efficient way to represent virtual world. It, however, remains challenging if entities of the virtual world such as 3-D models are to be transmitted or stored due to the large size of data. 3D Mesh Coding in this part of ISO/IEC 14496 is a way to efficiently compress a 3-D model with enhanced network capabilities: computational graceful degradation control, incremental representation, support for non-manifold model, error resilience, and quality scalability via hierarchical transmission of levels of details.

The Topological Surgery (TS) representation is the key element in coding of a 3-D mesh. A 3-D mesh can be thought of as IndexedFaceSet in ISO/IEC 14772-1:1997. And it consists of connectivity, geometry, and properties (e.g. colour, normal, texture coordinate) information. The connectivity information is encoded losslessly, whereas the other information could be quantised before compression. In order to maintain the congruence of the system, geometry and properties information are encoded in a similar fashion. The output of the decoder can be directly used as an IndexedFaceSet node represented in ISO/IEC 14496-1:2001.

O.2 Topological Surgery Representation

The connectivity of a 3D mesh is the relationship among the vertices, edges, and faces of the 3D mesh. In the Topological Surgery representation (TS) the connectivity of a connected component of a manifold 3D mesh is decomposed into *a Simple Polygon* and a *Vertex Graph*.

A non-manifold or non-orientable 3D mesh is first converted to an oriented manifold 3D mesh by cutting through some vertices and edges. One such method is described in reference [10]. The information necessary to reconstruct the original connectivity from the resulting orientable manifold connectivity can be encoded using *Stitches*.

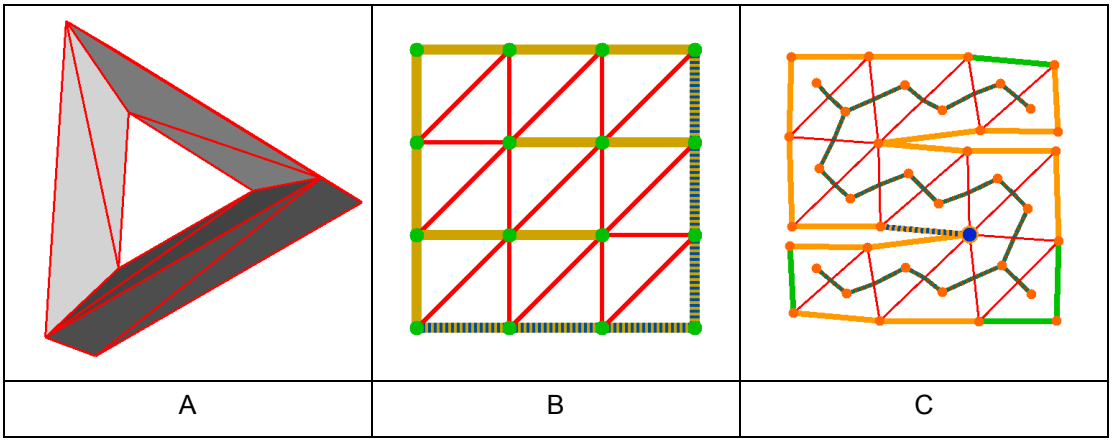


Figure O.1 — Example of Topological Surgery on a torus.
The torus (A) can be obtained from the quadrilateral (B) by identifying boundary vertices. The simple polygon is obtained by constructing a spanning tree in the dual graph of the mesh (C). The boundary of the simple polygon is the polygon loop. The vertex spanning tree is constructed as a spanning tree in the vertex spanning graph (B).

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

The first step of the encoding process, which generates this representation, is to construct a *face forest* spanning the dual graph of the 3D mesh. Two sets of the edges of the 3D mesh can be determined by the face forest: the *face forest edges* and the *vertex graph edges.* The vertex graph edges define the vertex graph spanning the graph of the 3D mesh. All the faces of each connected component of the 3D mesh are connected by face forest edges, resulting in one-to-one correspondence between connected components and trees of the face forest. The simple polygons are defined as the connected components of the 3D mesh resulting from cutting the original 3D mesh through the vertex graph edges. The decoder uses the vertex gr aph to identify how to stitch the 3D mesh back together. Figure O.1 shows an example of this decomposition.

The simple polygons are divided into triangle tree and triangle data pairs. This enables the decoder to incrementally render the model as data for each triangle is received from triangle data bitstream.

A non-manifold 3D mesh is represented as a manifold 3D mesh and a vertex clustering array. The vertex clustering array is represented as a series of stitches operations, with one command per vertex. The encoding process encodes the stitches after the vertex graph is encoded using the vertex clustering array, and optionally using a v_father array. The v_father array provides for each vertex in the vertex graph the index of its predecessor (father) in the directed vertex graph without jumps (potentially the vertex itself if it is a root of the graph without jumps). A pair (vertex, predecessor) is an edge of the 3D mesh.

In the actual representation to be described below, we force the face tree to be binary. We do this, by triangulating polygonal faces, effectively, adding additional edges to the mesh. Since this operation adds extra bits, we avoid unnecessary overheads in the vertex graph, by computing it previously to the triangulation.

O.2.1 Simple Polygon Representation

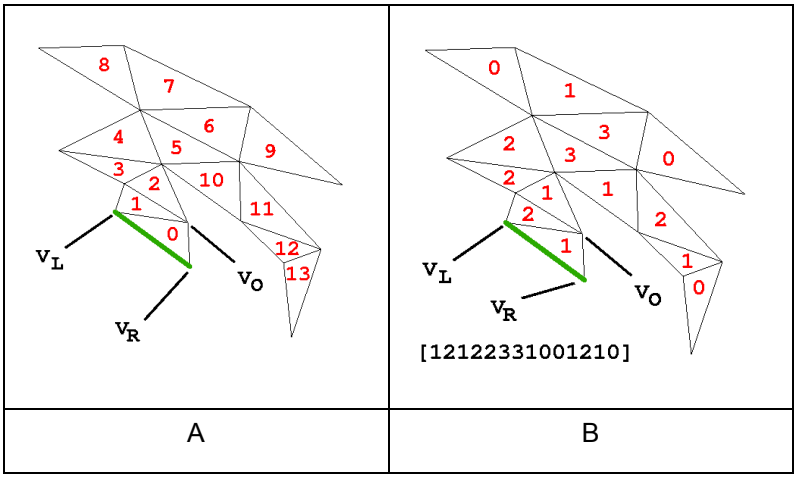


Figure O.2 — Constant-length encoding of a simple polygon. A: Triangles labels according to their order of traversal. B: Triangles labels according to their two bit code. The encoding of the polygon is the sequence between the brackets.

The simple polygon can be represented as a string of two-bit codes by recursively traversing the triangle tree as illustrated in Figure O.2. The traversal starts by entering a first triangle called *root triangle* through a polygon loop edge called the *root edge* (green edge in Figure O.2). One of the two endpoints of the root edge is called the *root vertex* of the simple polygon. The traversal algorithm uses three vertex pointers, the *left vertex* vL, the *right vertex* vR, and the *opposite vertex* vO. The right vertex is initialized to the root vertex, the left vertex to the other endpoint of the root edge, and the opposite vertex to the third vertex of the root triangle. The edge eL=(vL,vO) is called *left edge*, and the edge eR=(vR,vO) is called *right edge*. Each of the two edges is either a boundary edge of the polygon, or internal to the polygon. One bit of the encoding is used to indicate which case, with the bit associated with the left edge first followed by the bit associated with the right edge, and with values 1 for internal, and 0 for boundary. If only the left edge is internal, the traversal algorithm sets vR=vO and continues with the other triangle incident to eL. If only the right edge is internal, the traversal algorithm sets vL=vO and continues with the other triangle incident to eR. If both edges are internal, the traversal algorithm pushes vO and vR onto a traversal stack, it sets vR=vO and continues with the other triangle incident to eL. If both edges are boundary, and the traversal

,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

stack is not empty, the traversal algorithm pops vR and vL from the traversal stack, and continues with the other triangle incident to the edge (vL,vR). If both edges are boundary, and the traversal stack is empty, all the triangles of the simple polygon have been visited. For example, in Figure O.2-A, the triangles are labeled with their order of traversal, and in Figure O.2-B, with their corresponding two-bit code, as a number in the range 0,…,3. The encoding of this polygon is [12122331001210].

The simple polygon can also be represented as a *table of triangle runs* in the same way as the vertex tree, except that each triangle run record is composed of only two fields *(tt_run_length, tt_leaf).* A *Tlast* field is not necessary because the triangle tree is a binary tree, and each triangle run which ends in a branching node must be followed by exactly two runs. The structure of the triangle tree does not completely describe the triangulation of the simple polygon, though. To complete the description, an extra bit per triangle associated with a regular node of the triangle tree must be included. This *sequence of marching patterns* determines how to triangulate the triangle runs by advancing either on the left or on the right on the boundary of the simple polygon.

Note that there is a very simple transformation from the previous representation of the simple polygon to this one. The 3s and 0s in the sequence in the previous representation mark the end of the triangle runs. In the example of Figure O.2, the triangle-runs are defined by the subsequences [121223], [3], [10], [0], and [1210], which are in the same order defined by the table of triangle runs. The tt_run_length field value is the number of two-bit codes in the corresponding subsequence (6,1,2,1, and 4 in this case). The tt_leaf field value is determined by the last code in the subsequence ($3 \rightarrow 0$, $0 \rightarrow 1$), and the marching patterns by the other codes in the sequence, skipping the 3s and 0s ($1 \rightarrow 0$, $2 \rightarrow 1$). In this example the marching edge is the following sequence of bits [010110010]. Clearly, the opposite transformation is equally simple.

O.2.2 Vertex Graph representation

For a connected manifold without boundary, each edge of the vertex graph corresponds to two polygon loop edges (i.e., boundary edges of the simple polygon), but for manifolds with boundary, some edges of the vertex graph (those which correspond to boundary edges of the mesh) correspond to only one polygon loop edge. In the case of a *simple mesh* (manifold without boundary with Euler number 2, i.e., topologically equivalent to a sphere), the vertex graph is a tree. When the manifold mesh is not simple (i.e, when the mesh has boundary edges, or when the Euler number is different from 2), the vertex graph contains loops. If the vertex tree is constructed as a spanning tree by depth first traversing the vertex graph, then one back-edge, or *jump* edge, connects a leaf node of the vertex tree with a previously visited node, forming a loop.

The vertex graph is represented as data structure VG composed of sequence of *runs*. Even though each jump corresponds to a single edge in the vertex graph, we represent a jump by two runs in the VG, one of them having zero length. The first, and true, run starts at the first node the edge gets visited. The second run is generated when we reach the previously visited node, which, then, needs to be considered as a branching node, and the run generated has zero length. Each run record of VG is composed of a variable number of fields. A vertex run connects a leaf or branching node to another leaf or branching node through a path of zero or more regular nodes. The order of traversal of the tree defines a sequential order for the vertex runs, and a first and last node for each run. Each vertex run is represented as a vertex run record composed of the following fields. The vg_last field is a bit that determines if the run shares the first node with the next run or not. The vg_run_length field is an integer with a value equal to the number of edges in the run, not counting the jump edges. The vg_leaf field is a bit which determines if the run ends in a leaf or branching node. Each extended vertex run ending in a leaf node (vg_leaf=1) has an additional vg_loop field, indicating if the last node of the vertex run is connected to another node through a jump edge. Note that each extended vertex runs of length zero, which is always associated with a jump edge, should be considered as ending in a leaf. For these extended vertex runs, first and last node are the same. If vg_loop=1, the record also has a vg_forward_run field indicating whether this run corresponds to the start or end of the run. Each jump edge connects the last node of a run to a previously visited node. The node visited first is the *start* of the jump (vg_forward_run =1), and the other one is the *end* of the jump (vg_forward_run =0). If vg_forward_run =0, the record also has a vg_loop_index field. Every time the start of a jump is encountered, the compression and decompression algorithms push the reference to the jump edge onto an auxiliary *jump stack.* The value of vg_loop_index is the depth of the corresponding jump edge in the jump stack at the time the compression or decompression programs encounter the end of the jump. The reference is immediately removed from the stack, moving subsequent references up, and reducing the total depth of the stack by one (and so reducing the number of bits required to represent the depth of subsequent jump edges). Figure O.3 illustrates some of these concepts.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

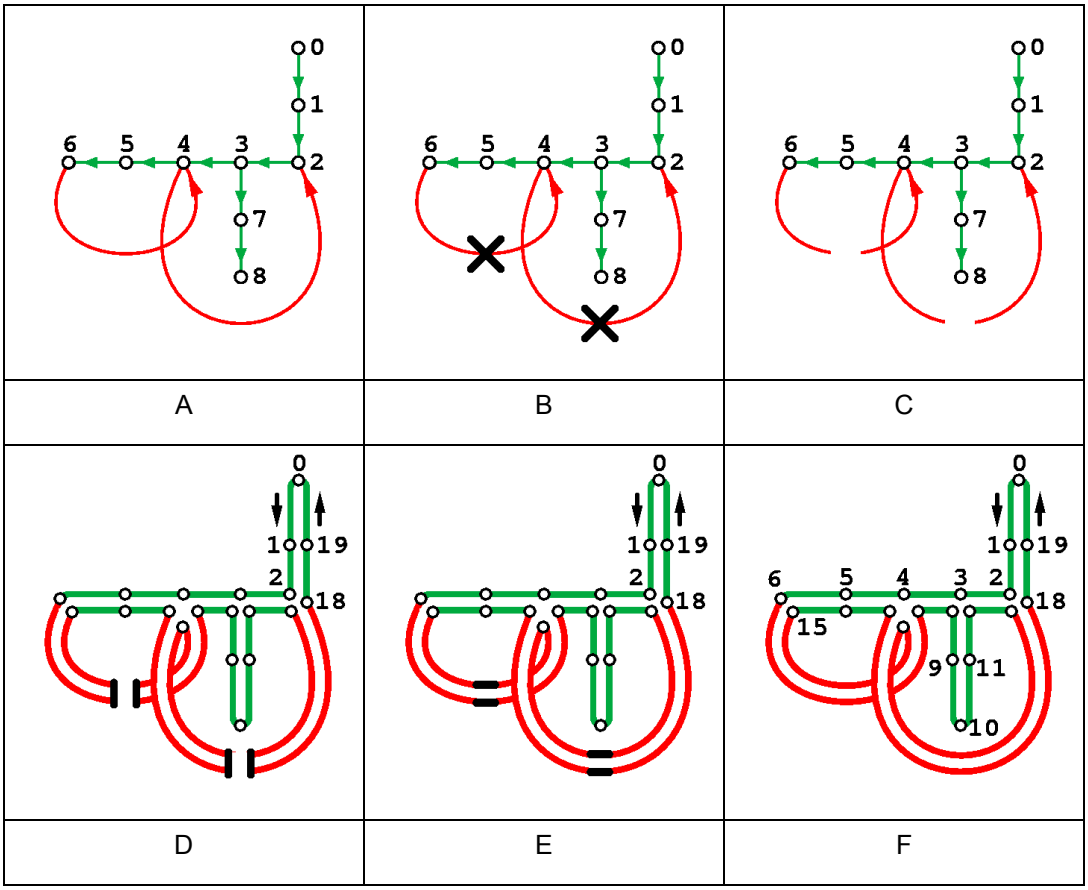


Figure O.3 — Steps to build the bounding loop from the vertex graph.
A : vertex graph decomposed into vertex tree (green) and jump edges (red).
B : extended vertex tree is created by cutting jump edges in half.
C : the extended vertex tree has two leaves for each jump edge.
D : build the extended vertex tree loop.
E : connect the start and end of each jump edge.
F : resulting boundary loop.

O.3 Encoding guidelines for 3D Mesh Coding

O.3.1 Topological Surgery Encoding

A manifold 3D mesh is represented in compressed form as the concatenation of its connected components represented in compressed form. The algorithm to compress a connected component of the connectivity of a manifold 3D mesh can be decomposed into the following steps:

1. Construct the face tree as a spanning tree in the dual graph of the 3D mesh.
2. Define the vertex graph as the graph resulting from deleting the face tree edges from the graph of the mesh.
3. Construct the vertex tree as a spanning tree in the vertex graph.
4. Label each edge of the 3D mesh as a polygon tree edge, a vertex tree edge, or a jump edge.
5. Triangulate the polygonal faces by adding internal non-polygonal edges to the faces.
6. The simple polygon is the mesh resulting from cutting the triangular mesh through the edges of the vertex graph. The triangle tree is the dual graph of the simple polygon.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7. Label each triangle tree edge as a polygon tree edge or non-polygon tree edge.

8. Choose an edge of the vertex tree and call it root edge of the vertex tree.

9. Choose an endpoint of the root edge of the vertex tree and call it root vertex of the vertex tree.

10. Of the faces incident to the root edge of the vertex tree, choose an incident face and call it root face of the face tree.

11. Call the edge of the root face of the face tree corresponding to the root edge of the vertex tree, root edge of the face tree.

12. Call the vertex of the root face of the face tree corresponding to the root vertex of the vertex tree, root vertex of the face tree.

13. Encode the simple polygon by depth first traversal starting at the root edge of the simple polygon, with the root vertex of the simple polygon as right vertex, and the other endpoint of the root edge as left vertex. Upon entering a branching triangle, traverse the left branch before the right branch.

14. Encode the vertex graph as a table of extended vertex runs by depth first traversing the extended vertex tree according to the orientation of the simple polygon induced by the order of traversal of the simple polygon.

This encoding procedure is described in further detail in [17].

O.3.2 Support for non-manifolds and Non-orientable manifolds

A non-manifold 3D mesh is represented as an orientable manifold mesh and a vertex clustering array. A non-orientable manifold 3D mesh is considered in the same way as a non-manifold with the inconsistent edges as singular. A first step in the encoding process consists of converting a non-manifold mesh to a manifold mesh and a vertex clustering array. There are several possible methods for performing this operation, one of which being described by Gueziec *et al.* in reference [10].

The Stitches Encoder operates in a manner symmetric to the Stitches Decoder. Its operation is summarized in the pseudo-code of Figure O.4 below.

```
encode(anchor_stack) {
  if (has_stitches == true) {
    encode has_stitches;
    for (i = nV0; i < nV1; i++) {
    // nV0 is the first vertex of the current component, and nV1 –1 is the last vertex
      encode stitch_cmd;
       if (stitch_cmd) {
          encode stitch_pop_or_get;
          if (stitch_pop_or_get) {
             encode stitch_pop;
             encode stitch_stack_index;
              retrieve stitching_anchor from anchor_stack;
             if (stitch_pop) {
                remove stitching_anchor from anchor_stack;
             } // end if
```


© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            encode stitch_incr_length;
            if (stitch_incr_length != 0) {
              encode stitch_incr_length_sign;
            } // end if
            decode stitch_push;
            if (stitch_push)
              push i to the back of anchor_stack
            retrieve stitch_length at stitching_anchor;
            total_length =  stitch_length + stitch_incr_length;
            if (total_length > 0)
             encode stitch_reverse;
            save total_length at stitching_anchor;
          } // end if (stitch_pop_or_get)
          encode stitch_length;
          push i to the back of anchor_stack;
      } // end if (stitch_cmd)
    } // end for
}
```

Figure O.4 — Pseudo-code for the Stitches encoder

If the boolean **has_stitches** flag in the current component is true, then for each vertex of the Vertex Graph in decoder order, the Stitches encoder encodes a stitching command from the bitstream. The boolean value **stitch_cmd** is encoded. If the boolean value **stitch_cmd** is true, then the boolean value **stitch_pop_or_get** is encoded; if the boolean value **stitch_pop_or_get** is false, an unsigned integer **stitch_length** is encoded. The current vertex i is then pushed to the back of the anchor_stack.

if **stitch_pop_or_get** is true, then the boolean value **stitch_pop** is encoded, followed with the unsigned integer value **stitch_stack_index**. Using **stitch_stack_index**, a stitching_anchor is retrieved from the anchor_stack. This is the anchor that the current vertex i will be stitched to. If the **stitch_pop** boolean variable is true, then the stitching_anchor is removed from the anchor_stack. Then,an integer **stitch_incr_length** is encoded as an unsigned integer. If it is different from zero, its sign (boolean **stitch_incr_length_sign**) is encoded. A **stitch_push** boolean value is encoded. If **stitch_push** is true, the current vertex i is pushed to the back of the anchor_stack. An integer **stitch_length** associated with the stitching_anchor is retrieved. A total_length is computed by adding **stitch_length** and **stitch_incr_length**; if total_length is greater than zero, then a **stitch_reverse** boolean value is encoded. Finally, the value total length is saved at (associated with) the stitching_anchor.

O.3.3 Support for Error Resilience

To support error resilience, data partitioning is very important. Component based data partitioning (CODAP) provides a new functionality to the TS method. CODAP provides a way to implicitly divide a large 3D mesh into multiple partitions. A partition may include part of a connected component or multiple connected components. Since vertex graph is relatively small compared to simple polygon, the division of connected component is only applied to simple polygon. However, the correspondence between vertex graph (vg) and simple polygon is kept through the whole coding process. There are three different partition types, which reflect different combinations of vertex graphs and simple polygons. In front of each partition, a start code is provided for resynchronization.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In order to make a partition into a given size, traversal order determination and partitioning methods are used. The traversal order information is coded with particular branching triangles. Virtual leaf triangle information is inserted at the end of each divided triangle tree to make a virtually complete triangle tree.

If channel errors hit the bit stream, some partitions will be corrupted or lost. However, other partitions will be decoded with its simple polygon and corresponding vg. Forward error correction may be used to ensure the vg is protected.

O.3.3.1 Traversal Order Determination

In TS, the reconstruction procedure is as follows: 1) building the bounding loop table, 2) computing the boundary offset for each branch, and 3) reconstructing triangles and polygons.

In basic TS coding, the traversal direction is always clockwise or left most search. In the case that the left branch is much larger than the right branch of the triangle tree, the decoder has to wait until it decodes the vertices all the way down the left branch before decoding and rendering the smaller right branch. See Figure O.5. However in the CODAP technique, in order to avoid unnecessary delay, the encoder will choose the shorter (left or right) branch. A one-bit flag will be sufficient to inform the decoder of the traversal order of the triangle tree.

In Figure O.5, the information of the traversal order is given at a branch on a main stem of the partitioned tree. The green branching triangle has the right first traversal order and the red branching triangle has the left first traversal order. Within a branch, the traversal order is always clockwise, as shown with the purple line.

me_1	me_2	…	td_orientation 1	...	me_i	$td_orientation_N$	…	me_n

The triangle data contains triangle marching edge (me) information where there is a triangle run. However, the triangle data contains the **td_orientation** fields where there is a branch in the main stem of the binary triangle tree.

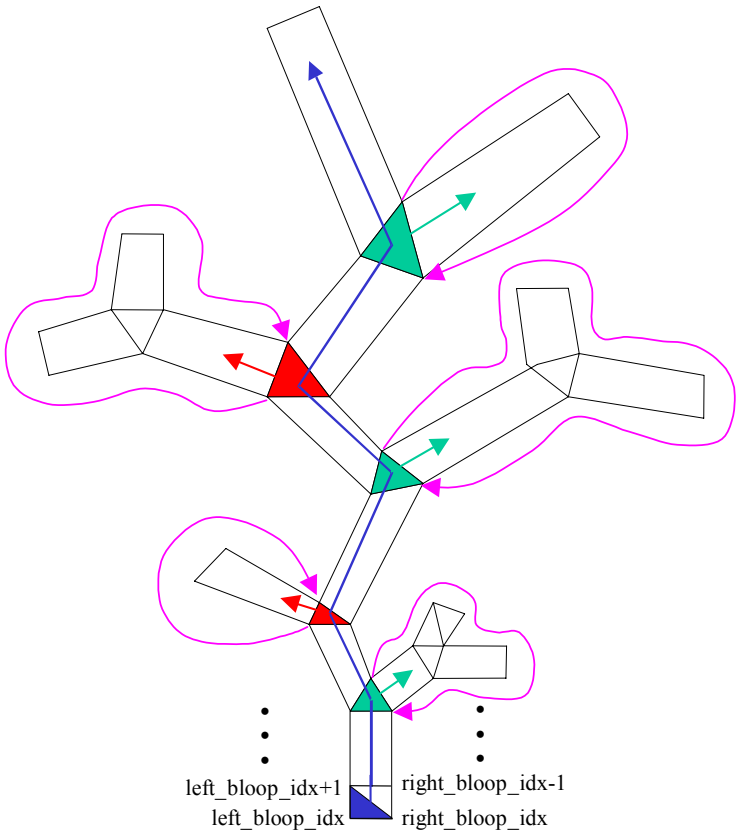


Figure O.5 — Determination of traversal order

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

O.3.3.2 Partitioning

One way to handle the Y-vertex while maintaining independence between partitioned simple polygon data is to make the partitioned data as a virtual connected component by inserting virtual bits, as shown in Figure O.6. There are two main cases for defining the virtual bits. When the partition ends with a run or a leaf triangle, a pair of virtual bits (1,1) is added to the (tt_run_length, tt_leaf) pair. When the partition ends on a branching triangle, two pairs of virtual bits (1,1) are added to the (tt_run_length, tt_leaf) pair. Also, if the partition ends on a branching triangle, the length of the branch (codap_branch_len) is coded so that Y-vertex index can be calculated.

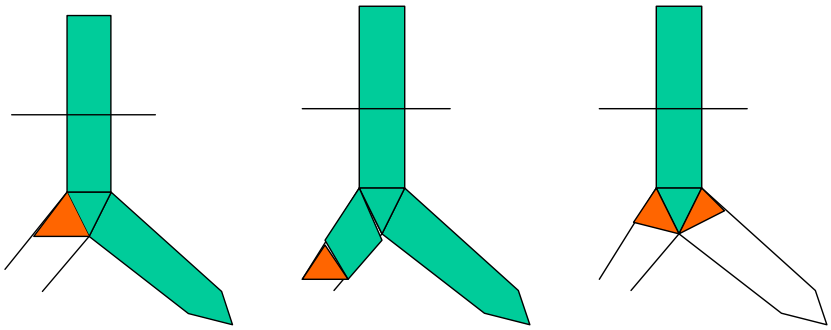


Figure O.6 — Examples of forming virtual connected component by inserting virtual bits to make virtual triangles (marked in red). The partitioning was done at : (a) a leaf (b) a run (c) a branch.

When the simple polygon data is partitioned and formed as virtual connected components, the decoder can find out if the virtual bits exist by checking the following condition:

codap_right_bloop_idx – codap_left_bloop_idx – 1 > total number of triangles decoded in tt.

If this condition is true, then virtual bits exist. To find out if there are one or two pairs of virtual triangles, the third triangle from the end is checked to see whether it is a branch or not. If it is a branch, then the partition ended on a branch and two virtual triangles are added. Otherwise, the partition ended on a leaf or a run and only one virtual triangle is added at the end. However, if the leaf is the only triangle in the subtree in Figure O.6 (a), encoders shall not make a partition at this point. This is to avoid a conflict with Figure O.6 (c).

Also the extended boundary prediction mode shall be used for the vertices in the partition of the subtree unless every vertex in the partition of the subtree is not used in the partitions after the subtree.

In topological surgery, to code the mesh information containing polygons, the polygons are first divided into triangles. Figure O.7 shows an example for this case.

--`,,`,,`-`-`,,`,,`,`,,`---

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

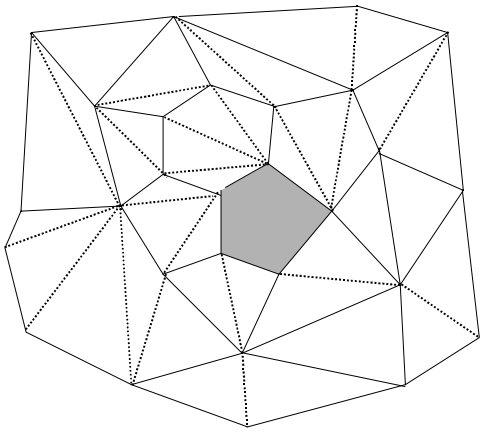


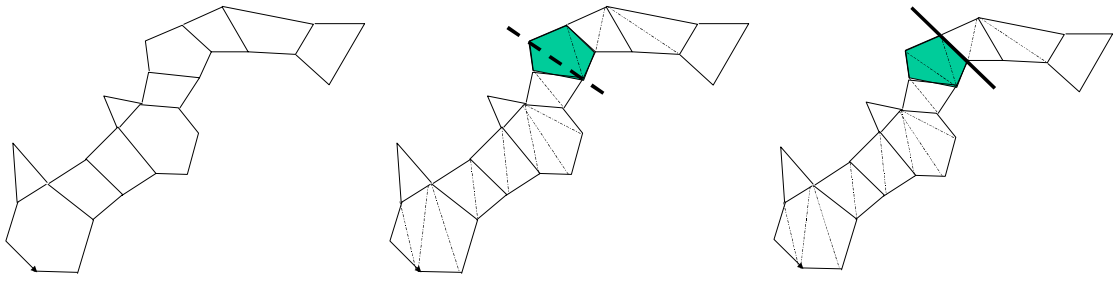
Figure O.7 — Example of dividing 3D mesh into triangle mesh : the solid lines are the original 3D mesh edge, and the dotted lines are the imaginary edges added for the division of polygon into triangles.

When polygons are divided into triangles, imaginary edges are added. Therefore, this information must be sent to the decoder to reconstruct the polygons. This information is called polygon edge. So when a polygon edge information is sent for each triangle, value 1 is sent for a real edge, and 0 is sent for an imaginary edge. However, if 3D mesh does not exist (only triangle meshes : triangulated==1), then polygon edge information is not needed.

In the topological surgery method, every polygon in the mesh is divided into triangles and polygon edge information is produced by the number of triangles. All of this information is coded, transported and stored. However, since the first triangle must start with a real edge, the corresponding polygon information will always be 1, and so it is not coded. In this case, the decoder assumes that the polygon edge information for the first edge is 1.

When data partitioning is performed, the partitioning may be done at the imaginary edge inside the polygon, as in Figure O.8 (b). In this case, it cannot be assumed that the polygon edge of a data partition starts with 1. Therefore the polygon edge must be coded by the number of triangles, except for the simple polygon data in partition type 0, which is described in the next section.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



(a) (b) (c)

tt_1	tt_2	…	tt_n	me_1	me_2	…	me_n

(d)

tt_1	tt_2	…	tt_n	me_1	me_2	…	me_n	pe_2	pe_3	…	pe_n	…

(e)

tt_1	tt_2	…	tt_n	me_1	me_2	…	me_n	pe_1	pe_2	pe_3	…	pe_n	…

(f)

Figure O.8 — Example of partitioning on 3D mesh and the related syntax : tt is the triangle run information, me is the marching edge information, pe is the polygon edge information, annex at the bottom shows the order of tm and pe corresponding to tt, n is the number of triangles. (a) the original 3D mesh, (b) partitioning inside a polygon, (c) partitioning at the boundary edge(real edge) of two polygons, (d) when triangulated == 1(no 3D mesh) (e) when 3D mesh exists and partitioning is only allowed at the real edge, (f) when 3D mesh exists and partitioning is allowed at the imaginary edge.

On the other hand, if there exists a polygon with many sides (edges) but it is partitioned only at a real edge, the size of a partition may not come into the range of target size. So when the partition inside the polygon is allowed, it is possible to achieve the size that within a small range of target size. Also, allowing the partition inside the polygon makes the method more generalized.

O.3.3.3 Boundary Prediction

Once a connected component is partitioned into pieces, prediction of geometry coding (as well as other attributes that use predictive coding) should allow independent decoding between partitions.

There are two modes of coding vertices on the boundary between partitions: restricted boundary prediction mode and extended boundary prediction mode.

Confer the details of both modes in subclause 7.13.5.3

When partitioning is done at a branching triangle, it is recommended to use extended boundary prediction mode for the first branch from the branching triangle.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

O.4 Encoder considerations for efficient compression of Vertex Properties

Given a geometry sample (x, y, z), the quantised geometry sample (qx,qy,qz) is obtained as:

xq = floor((x – **coord_xmin**)/**coord_size***((1<<**coord_quant**)-1)+0.5)

yq = floor((y – **coord_ymin**)/**coord_size***((1<<**coord_quant**)-1)+0.5)

zq = floor((z – **coord_zmin**)/**coord_size***((1<<**coord_quant**)-1)+0.5)

The procedure for colors and texture coordinates is the same.

A higher-order geometrical prediction scheme is employed in coding of vertex positions, color properties, texture positions, and normal vectors. The encoder transmits **pred_type,** the information of prediction type, to the decoder, as shown in Table O.1.

Table O.1 — Meaning of pred_type

pred_type	Prediction type
00	No_prediction
01	Tree_prediction
10	Parallelogram_prediction
11	reserved

For example, if the **pred_type** indicates the prediction type is tree-prediction or parallelogram, the predicted value($\tilde{p}$) can be expressed as shown in Eq. O.x.1.

$$\tilde{p} = \sum_{i=1}^{N} \lambda_i p_i \qquad \text{(Eq. O.x.1)}$$

The p_i's in Eq. O.x.1 represent the vertex properties of the adjacent vertices. The encoder transmits the each λ_i and *N* to the decoder.

However, the vertex property values such as colors, positions, normal vectors, and texture positions along the vertex traversal sequences has low correlation between successive vertices, it degrades the prediction accuracy. In the case of low correlation sequences, the prediction scheme can be limited to the simple 1st-order prediction. The simple 1st-order prediction scheme can be easily implemented to the encoder and decoder by fixing the λ_i's, as shown in Eq. O.x.2.

$$\lambda_i = \begin{cases} 1 & \text{, where } \quad i = 1 \\ 0 & \text{, otherwise} \end{cases} \qquad \text{(Eq. O.x.2)}$$

Table O.2 shows the weighting factors of the prediction schemes. The higher-order prediction mode is the general parallelogram predictor, while the simple 1st-order prediction mode is the modified encoding scheme. The simple 1st-order prediction mode shows better performance for color property coding.

Table O.2 — weighting factors of predictor

Prediction scheme	N	λ_i	Coding efficiency
parallelogram	3	1, 1, -1	-
Simple 1st-order prediction	L (> 0)	1, 0, … , 0	better for color encoding

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

O.5 Progressive Forest Split Representation

In the progressive forest split (PFS) compression scheme, a 3D model is represented as a low resolution model followed by a sequence of forest split operations. A forest split operation is represented by a forest in the graph of vertices and edges of a mesh, a sequence of simple polygons, and a sequence of vertex and property displacements. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with one of the simple polygons, and finally, displacing the new vertices and properties. The representation, encoding scheme, simplification algorithm, encoding and decoding methods are described in [8].

The general structure of a simplification scheme to generate the progressive representation, and the associated level of detail hierarchy, from a single resolution 3D mesh, is described as well. This method may require additional compression parameters, which may be tuned interactively for each transition from one level of detail to the next one.

O.5.1 Encoding the Forest

A simple encoding of the forest requires one bit per edge, for example a value 1 for the edges belonging to the forest and 0 for the rest. These bits are placed in the compressed data stream in the edge index order defined above. However, since any subset of edges of a forest form a forest, the edges with bit value 1 read up to a certain point may determine that certain edges with higher edge index should have bit value 0, otherwise they would create loops in the forest defined so far by the bits with bit value 1. These edges can be skipped in the compressed data stream. Arithmetic encoding of the bit-stream results in fewer bits. In the experiments that we have run so far, where the number of triangles increase by 50-70% with each forest split operation, the skipping of predictable bits described above trims the length of the forest bit-stream by about 1-8%, and the simple encoding of the resulting bit-stream is usually shorter than the run-length encoded stream. A pseudo-code implementation of the method to determine which edges to encode is as follows:

```
/ forest encoder
// marked edges form a forest in the graph of the mesh

int nV = number of vertices of mesh
Partition p(nV)
// p defines a partition of a set of nV elements
// initially each part contains a single element
// the method find(i) retrieves the number of the part containing element i
// the method join(i,j) joins the parts containing elements i and j, respectively

for ( every edge e={i,j} of mesh ) {
  bool skip_edge = false;
  bool edge_bit  = (e.isInForest)?true:false;
  if(p.find (i)!=p.find(j)) {
    if(edge_bit==true)
      p.join(I,j);
  } else {
    // forest => edge_bit==false
    skip_edge = true;
```

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  }
  if(skip_edge==false)
    encode_bit(edge_bit);
}
```

Figure O.9 — forest encoder pseudo-code

O.5.2 Support for meshes with polygonal faces

The decoder process supports polygonal faces for the base mesh, as well for the simple polygons used to fill the tree loops.

O.5.3 Method for generating of a PFS Representation of a Triangular 3D Mesh

The method developed in this section takes as input a 3D mesh, where all polygons are triangles and generates a series of levels of detail that can be encoded as a series of PFS operations. The implementation included in the reference software takes as an input a restricted VRML single-resolution file format (described in the Core Experiments document) and returns a Clustered multi-resolution file format (extension ".cmr" as described in the Core Experiments document). Although a Clustered multi-resolution file format can represent any hierarchy of levels of detail, the output of our method is a particular hierarchy corresponding to a base mesh and a succession of PFS refinements. The method is based on applying a succession of edge collapses to the original mesh. The edge collapse operation is described in the Core Experiments document, in the section dedicated to the Progressive Vertex Split method: it consists of merging the two endpoints of an edge, and thereby eliminating two triangles from the mesh (one triangle if the edge belongs to the mesh boundary). The method of the reference implementation consists of entering all the edges of the mesh in a priority queue that is keyed with the length of the edge. Then, in succession the edges are removed from the queue, and collapsed provided that they pass a series of tests. The tests are of two types: "topological" tests and "geometric" tests. Topological tests are performed first, as they are expected to be less costly than the geometric ones. If the tests are all passed, the edge is removed from the queue, and all neighboring edges are removed and re-entered in the queue that should reflect their new lengths. Before the geometric tests are performed a decision is made on how to position the vertex resulting from the edge collapse. Depending on the user's choice, this position can be either edge endpoint or the midpoint of the edge. In the sequel, the vertex resulting from the edge collapse is referred to as "simplified vertex". Figure O.10 is a diagram illustrating the algorithm.

```
while queue not empty
        take shortest edge
        if edge can be collapsed, e.g.,
                (valence does not exceed maximum &&
                 mesh is still manifold after operation &&
                 all collapses correspond to pfs &&
                 triangle rotation is acceptable &&
                 triangle compactness is within bounds)
           remove all edges in the star from the queue
           insert new edges
        else /* collapse of edge is not possible */
           simply remove edge from queue
```

Figure O.10 — Algorithm for generating PFS representation by collapsing edges of a 3D mesh

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

O.5.4 Topological Tests

The purpose of the topological tests is to ensure that the mesh is still a manifold after collapsing a given edge and that the collection of triangles that were removed are such that the dual graph of this collection is a forest. The first test verifies that the valence of the vertex resulting from the edge collapse, or simplified vertex, is at east 3 if it is not a boundary vertex. The second test ensures that all the vertices incident to the simplified vertex are different (otherwise we would have a non-manifold mesh). The third test verifies that the collection of triangles that were removed by collapsing edges form a forest as well as a manifold mesh. Both properties are checked by means of a comprehensive test. To keep track of the triangles that are eliminated, we introduce a color coding of the triangles as follows: blues triangles are the ones eliminated during edge collapses; red triangles are the ones remaining, purple triangles are the triangles that would be eliminated (would become blue) if the edge collapse under current testing would be performed. After several collapses are performed, an edge in the simplified surface can correspond to a pair of non-adjacent edges in the original surface (this is quite easy to see, for instance when collapsing in sequence two adjacent edges). It follows that the triangles that are removed during the collapse (purple triangles) are not necessary adjacent. Two cases are distinguished, depending on whether the purple triangles are adjacent or not. In the first case, the two purple triangles, which we refer to as T0 and T1, are adjacent. We use the notations of Figure O.11. We first test whether T0 and T1 are adjacent to 3 or more blue triangles, in which case the test fails. Otherwise, if T0 and T1 are adjacent to 0 or 1 blue triangle, we consider the vertices among V0, V1, V2 and V3 (shown in Figure O.11) that are not incident to a blue triangle that is adjacent to either T0 or T1. If there are incident to any blue triangle the test is failed (Figure O.11B), and otherwise the test is passed. Otherwise, if T0 and T1 are adjacent to exactly two blue triangles, we first check whether the two adjacent blue triangles are incident to a common vertex, if which case the test is failed. Otherwise, the two adjacent blue triangles must belong to a different connected component of blue triangles in order to pass the test (Figure O.11A). The connected components of triangles are maintained using a partition of the triangles.

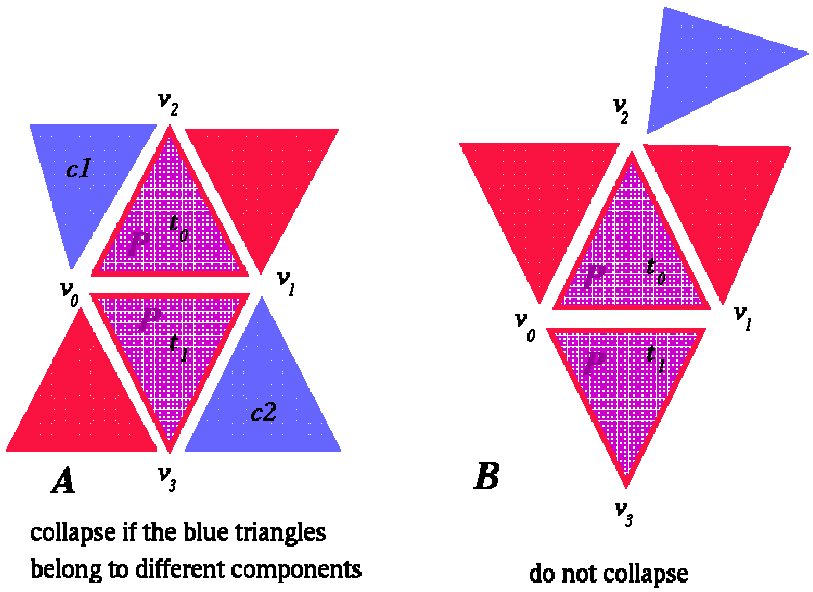


Figure O.11 — Additional edge collapsibility tests necessary to obtain a PFS representation. The cases illustrated here apply when the two triangles of the original surface that are removed by the edge collapse are adjacent

The second case applies when the two purple triangles T0 and T1 are not adjacent. This case is illustrated in Figure O.12. We first test whether Vertex V2 is a boundary vertex. If yes, T0 must be a boundary triangle otherwise the test fails (Case A in). If T0 is a boundary triangle, then the test is the same as when V2 is not a boundary vertex (Cases B and C in Figure O.12): all triangles incident on V2 must be red. The same sequence of tests is applied to Vertex V4: if Vertex V4 is a boundary vertex, T1 must be a boundary triangle otherwise the test fails (Case A in). If T1 is a boundary triangle, then the test is the same as when V4 is an interior vertex (Cases B and C in): All triangles incident on V4 must be red.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

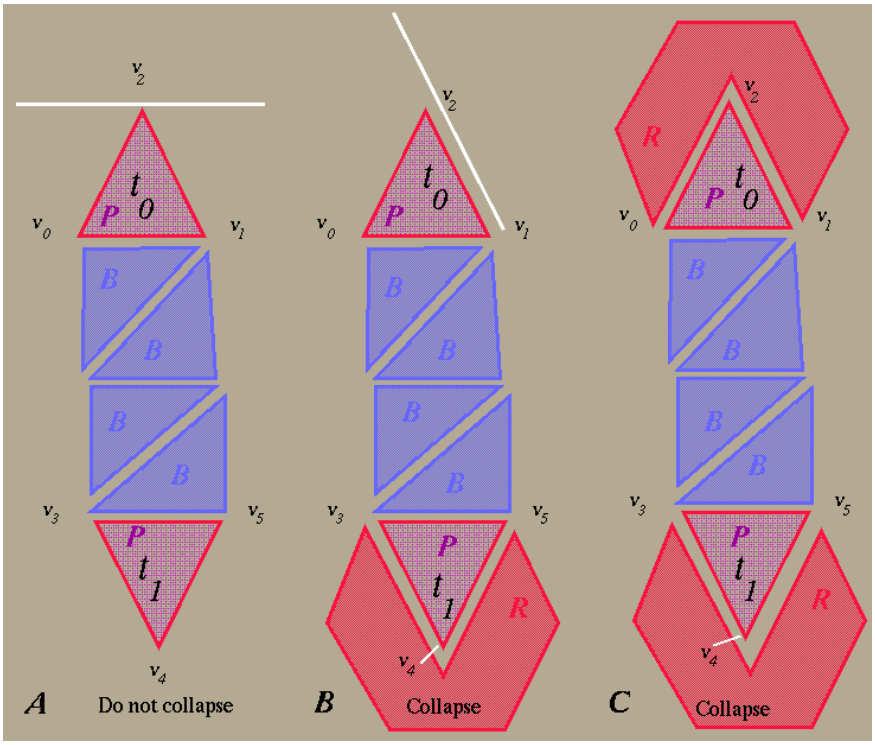


Figure O.12 — Additional edge collapsibility tests necessary to obtain a PFS representation. The cases illustrated here apply when the two triangles of the original surface that are removed (the purple triangles T0 and T1) are not adjacent. Boldface letters inside triangles indicate colors. A: do not collapse. B, C: collapse.

O.5.5 Geometric Tests

The geometric tests described in this section are not compulsory in order to represent the level of detail hierarchy in PFS. However, they were experimentally useful to produce levels of detail that are visually satisfactory. This part of the algorithm is most open for changes and improvements by parties implementing an encoder. The first geometric test of the sample implementation verifies that the normals of the triangles incident to any vertex of the collapsed edge do not rotate by more that a user-specified angle during the collapse. The user can specify this maximum angle. The second geometric test verifies that the minimum compactness values of the triangles incident to any endpoint of the collapsed edge, before the edge collapse (C0) and after the edge collapse (C1) satisfy the following relation: C1 > r C0, where r is a user-specified ratio. The compactness of a triangle is defined using the above formula, $c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2}$, where *a* is the area of the triangle and the l_i are the three side lengths.

O.6 Complexity estimation for Computational Graceful Degradation

Computational Graceful Degradation (CGD) is the process of gracefully degrading the quality of the rendering of the 3D scene in case the 3D rendering engine has not enough processing power for performing the rendering in real-time. In this case, some lower-quality rendering processes can be performed instead, as for example flat shading instead of smooth shading, mesh-rendering with a smaller number of triangles, … The implementation of specific CGD algorithms which provide the highest complexity reduction for the lowest image degradation is an implementation dependent issue which has no normative implications.

For being able to apply Computational Graceful Degradation, each specific 3D rendering engine should be able to estimate its required rendering processing power, before the 3D rendering is actually started. The encoder must therefore transmit, as soon as possible, some complexity estimation (ce) characteristics of the 3D objects, which are relevant to 3D rendering process. The definition of these complexity parameters is normative. These parameters are defined as follows:

--`,,-`-`,,`,,`,`,,`---
© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1. Number of vertices

2. Number of triangles

3. Number of edges

4. Projected surface of the object in any viewpoint

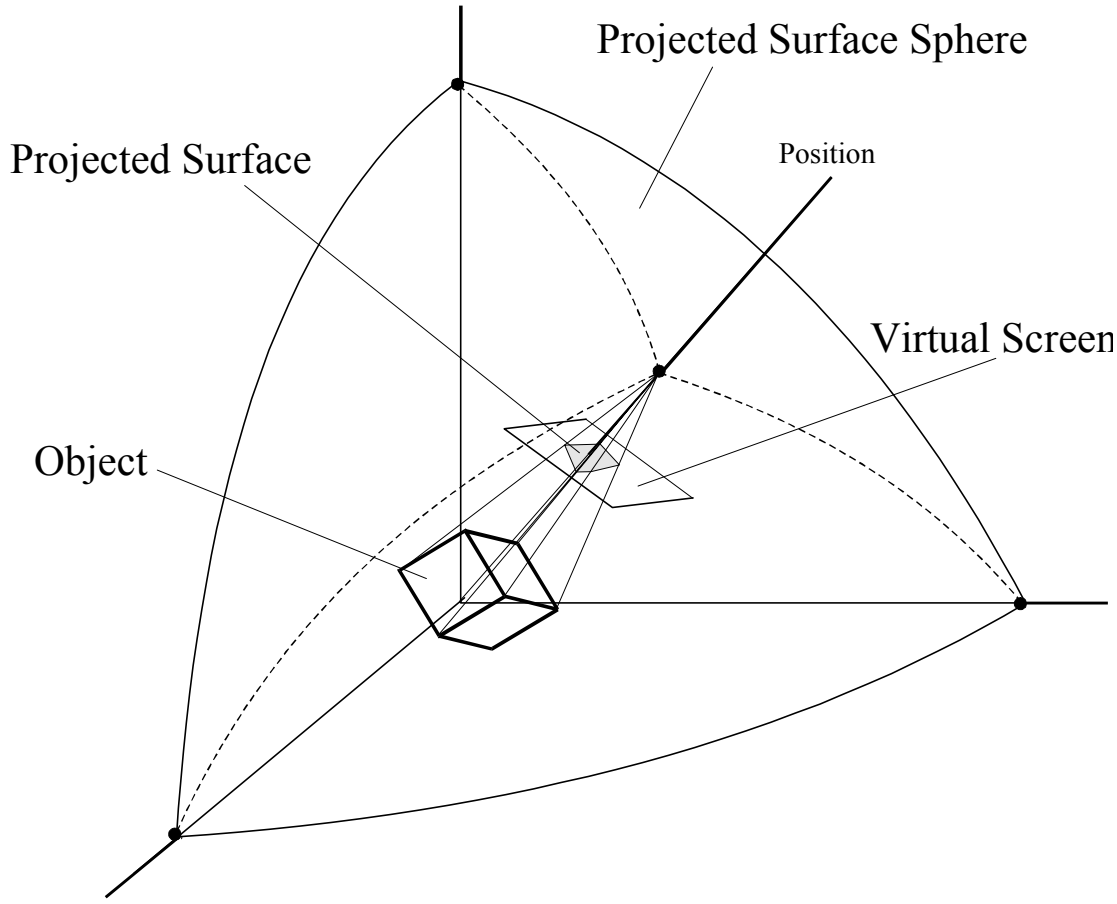


Figure O.13 — Projected Surface Sphere constructed by projecting the pixels of the object onto a virtual screen for different viewing angles

The projected surface is the number of pixels that are actually projected onto the screen (called the virtual screen) during the rendering of the object in that viewing position (see Figure O.13). This projected surface largely varies with orders of magnitude, depending on the viewing angle and viewing distance to the object, which can be modified by the user, without any encoder interaction. In principle, the projected surface should be transmitted:

— for any position around the object, with constant distance to a reference point of the object (e.g. gravity point), creating the so-called Projected Surface Sphere (see Figure O.13).

— for one particular viewing angle, in a variety of distances to the reference point.

However, simulations have shown (see [9]) that the projected surface should only be measured in a limited number of positions around the object, from which the projected surface for any other position can be recovered - within an acceptable error bound - by linear interpolation or scaling. Linear interpolation is used within a Projected Surface Sphere with constant radius, while scaling is required for extrapolating (interpolating) information from one (different) Projected Surface Sphere(s) to another Projected Surface Sphere. For better accuracy, we advise to apply the following scaling formula, using one particular Projected Surface Sphere:

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$p_i = \frac{R_{ref}}{R_i}.\left(p_{ref} - \min_{ref}\right) + \min_i$$

in which:

p_i is the estimated projected surface in a specific direction for radius R_i.

p_{ref} is the measured projected surface in that specific direction for radius R_{ref}.

R_{ref} is the radius of the reference Projected Surface Sphere.

R_i is the radius of the estimated Projected Surface Sphere.

min_{ref} is the minimum projected surface for the reference Projected Surface Sphere.

min_i is the minimum projected surface for the estimated Projected Surface Sphere.

The accuracy can even be increased by allowing some additional measured Projected Surface Spheres (thus at different radii). Typically, only a very limited points will be added for each additional Projected Surface Sphere.

A 1-bit switch should indicate in the 3D_Mesh_Object_Header whether Computational Graceful Degradation is enabled or not. This bit is called the ce_SNHC. If the ce_SNHC is set to off, no additional information with respect to complexity estimation is sent through the bitstream. Otherwise, the Projected Surface Sphere(s) is (are) sent, from which the required processing power for 3D rendering of the 3D object can be estimated. Since the projected surface does not drastically change with the 3D mesh resolution, it is specified in the 3D_Mesh_Object_Header for all layers, instead of being specified in 3D_Mesh_Object_Layer.

For each 3D_Mesh_Object_Layer, information regarding the number of vertices, triangles and edges should be transmitted if the ce_SNHC is enabled in 3D_Mesh_Object_Header. Transmitting the number of vertices, triangles and edges at the level of the 3D_Mesh_Object_Layer has the advantage that the 3D rendering estimation can be performed at any 3D mesh resolution.

O.7 QoS for SNHC through upstream

Defining profiles and levels in SNHC has proved to be considerably difficult, since processing power required for decoding and rendering the 3D content varies with orders of magnitudes in time, while additionally the performance of decoders is highly dependent on the target platforms (software versus hardware implementations, very powerful graphics cards, …).

Figure O.14 shows how this can be resolved by using Computational Graceful Degradation techniques in the decoder.

However, this does not resolve the problem of profile and level definitions, in which basically the data that is transmitted should comply to some predefined thresholds (equivalent to “compliant bitstreams”). Therefore, we provide an upstream, which syntax conveys some information that reveals to the encoder (server) the performances of the decoder (see Figure O.14). This basically means that the decoder determines its capabilities in advance (one-shot calibration process), transmits this information to the encoder, which will “restrict” its data to the capabilities of the decoder (e.g. by mesh simplification). The data to be transmitted through the upstream is defined in subclause 6.3.12.3 (upstream_SNHC_QoS).

The decoder performances are measured by decoding and rendering different SNHC objects and measuring the frame rate in different configurations, i.e. a varying number of triangles, a varying number of pixels projected on the screen and different rendering modes. The performance information is transmitted using different performance curves. Each performance curve corresponds to the frame rate achieved for one particular setting of the number of triangles and rendering mode, but for a varying number of screen coverages.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

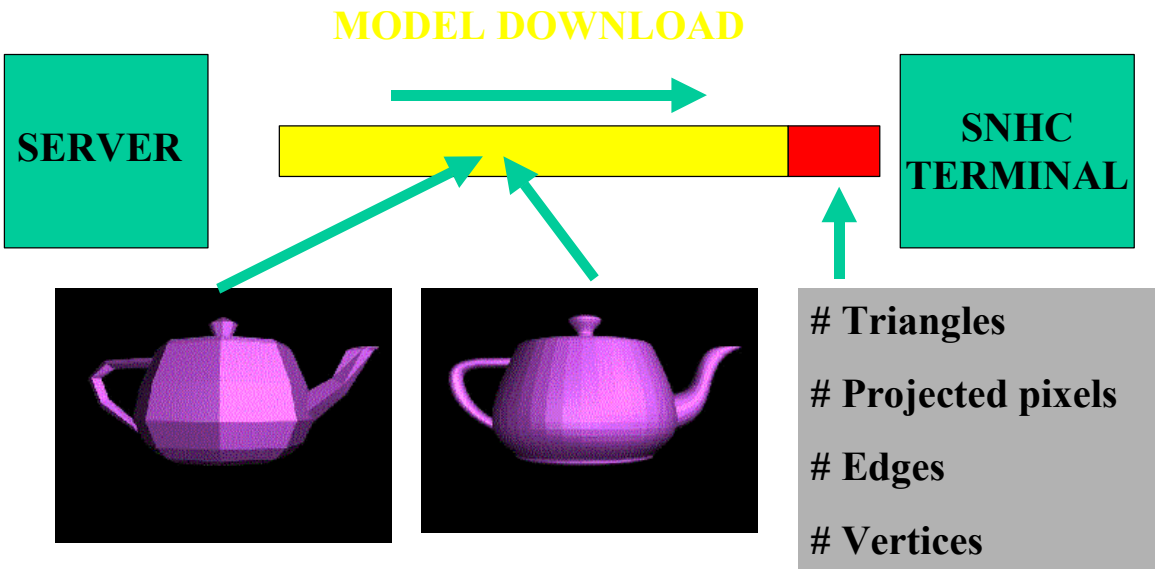


Figure O.14 — Downloading complexity estimation (Computational Graceful Degradation) parameters to the decoder

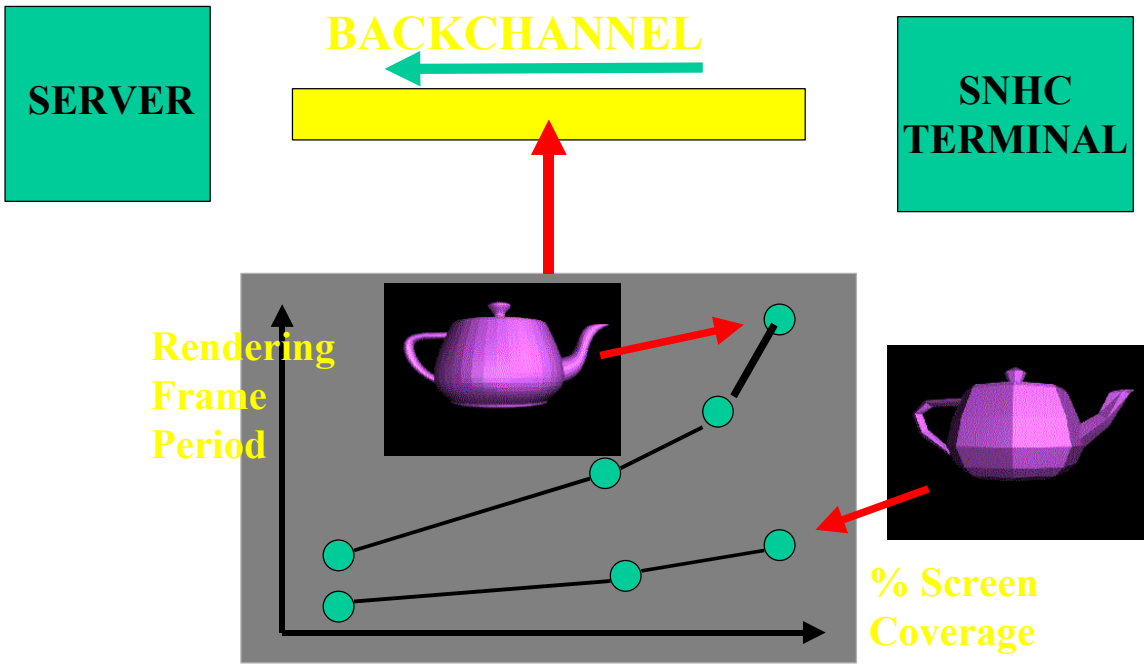


Figure O.15 — Uploading terminal (renderer) performance figures to the server, which will in turn transmit the data model at the correct Level of Detail for enabling real-time rendering at the decoder

O.8 References

[1] W.B. PENNEBAKER and J. MITCHELL, "*JPEG Still Image Data Compression Standard*," New York: Van Nostrand Reinhold, 1993

[2] JIANKUN LI, JIN LI and C.-C. JAY KUO, "An Embedded DCT Approach to Progressive Image Compression, ICIP'96", *IEEE International Conference on Image Processing*, Vol. 1

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[3] David TAUBMAN and Avideh ZAKHOR, "Multirate 3D Sub-band Coding of Video", *IEEE Transactions on Image Processing*, Vol 3.,No. 5, September 1994

[4] G. TAUBIN, "A Signal Processing Approach To Fair Surface Design", *Siggraph'95 Conference Proceedings*, pp 351-358, Los Angeles, August 1995

[5] G. TAUBIN and J. ROSSIGNAC, "Geometry Compression through Topological Surgery", *ACM Transactions on Graphics*, pp 84-115, April 1998

[6] G. TAUBIN, W.P. HORN, F. LAZARUS and J. ROSSIGNAC, "Geometric Coding and VRML", *Proceedings of the IEEE, Special Issue on Multimedia Signal Processing*, July 1998

[7] G. TAUBIN, W.P. HORN and F. LAZARUS, "*The VRML Compressed Binary Format*", 1997. http://www.research.ibm.com/vrml/binary

[8] G. TAUBIN, A.GUEZIC, W.P. HORN and F. LAZARUS, "Progressive Forest Split Compression", *Siggraph'98 Conference Proceedings*, pp. 123-132, Orlando, July 1998

[9] Gauthier LAFRUIT, Lode NACHTERGAELE, Kristof DENOLF and Jan BORMANS, "3D Computational Graceful Degradation", *The 2000 IEEE International Symposium on Circuits and Systems*, Geneva, Switzerland, May 2000

[10] A. GUEZIEC, G. TAUBIN, F. LAZARUS and W.P. HORN, "Converting Sets of Polygons to Manifold Surfaces by Cutting and Stitching," in *Visualization'98 Conference Proceedings*, Research Triangle Park, NC, October 1998

[11] A. GUEZIEC, G. TAUBIN, F. LAZARUS and B. HORN, "A Framework for Streaming Geometry in VRML", *IEEE Computer Graphics & Applications*, special issue on VRML, March-April 1999

[12] A. GUEZIEC, F. BOSSEN, G. TAUBIN and C. SILVA, "Efficient Compression of Non-Manifold Polygonal Meshes", IEEE Visualization'99, 1999

[13] A. GUEZIEC and G. TAUBIN, "A Framework for Memory-Efficient Levels of Detail", *Tenth Canadian Conference on Computational Geometry*, Montreal, Canada, August 1998

[14] G. TAUBIN and J. ROSSIGNAC, "Course on 3D Geometry Compression", *Siggraph'98*, Orlando, FL, July 1998

[15] G. TAUBIN and J. ROSSIGNAC, "Course on 3D Geometry Compression", *Siggraph'99*, Los Angeles, CA, 1999

[16] G. TAUBIN, W. HORN and P. BORREL, "Compression and Transmission of Multi-Resolution Clustered Meshes", *IBM Technical Report* RC-21398, February 1999

[17] F. BOSSEN, "*On the Art of Compressing Three-dimensional Polygonal Meshes and their Associated Properties*", Thesis no. 2012, Ecole Polytechnique Federale de Lausanne, September 1999

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] Arun N. NETRAVALI and Barry G. HASKELL, "*Digital Pictures, representation and compression*", Plenum Press, 1988

[2] Barry G. HASKELL, Atul PURI and Arun N. NETRAVALI, "*Digital Video: An Introduction to MPEG-2*", Chapman & Hall, ISBN 0-412-08411-2, 1997

[3] F. I. PARKE and K. WATERS, *Computer Facial Animation*, A K Peters, Wellesley, MA, USA, 1996

ISO/IEC 11172-1:1993, *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 1: Systems*

ISO/IEC 11172-3:1993, *Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s — Part 3: Audio*

ISO/IEC 13818-3:1998, *Information technology — Generic coding of moving pictures and associated audio information — Part 3: Audio*

IEC 60461:1986, *Time and control code for video tape recorder*

ITU-T Recommendation T.81 (1992) | ISO/IEC 10918-1:1994, *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex P
(normative)

Data mapping format of SMPTE12M (time code)

The 64bit contents of time_code_smpte12m in Group of Studio Video Object Plane header and Studio Object Plane header is mapped as follows:

Syntax	Bits	Mnemonic
Time_code[63..0]{		
Color Frame Flag	1	bslbf
Drop Frame Flag(NTSC)/Unused(PAL)	1	bslbf
TV Frame Tens	2	uimsbf
TV Frame Units	4	uimsbf
Field Phase(NTSC)/Binary group flag 0(PAL)	1	bslbf
TV Seconds Tens	3	uimsbf
TV Seconds Units	4	uimsbf
Binary group flag 0 (NTSC)/Binary group flag 2(PAL)	1	bslbf
TV Minutes Tens	3	uimsbf
TV Minutes Units	4	uimsbf
Binary group flag 2 (NTSC)/ Field Phase (PAL)	1	bslbf
Binary group flag 1 (NTSC)/Binary group flag 1(PAL)	1	bslbf
TV Hours Tens	2	uimsbf
TV Hours Units	4	uimsbf
2nd Binary Group	4	uimsbf
1stBinary Group	4	uimsbf
4th Binary Group	4	uimsbf
3rd Binary Group	4	uimsbf
6th Binary Group	4	uimsbf
5thBinary Group	4	uimsbf

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8th Binary Group	4	uimsbf
7thBinary Group	4	uimsbf
}		

NOTE If the definition of SMPTE12M is modified, time code in this specification shall also be modified in the same manner.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex Q
(informative)

Syntax correspondence of MPEG-2 4:2:2 Profile to MPEG-4 Studio Profile

This informative annex describes syntax correspondence of the MPEG-2 4:2:2 Profile to the MPEG-4 Studio Profile. This Annex provides useful information for bitstream translation.

Q.1 Sequence header

MPEG-2 4:2:2 Profile syntax	***MPEG-4 Studio Profile***		***Note***
	layer	***syntax***	
horizontal_size_value	StudioVideoObjectLayer()	**video_object_layer_width**	MPEG-2:12bits + 2bits (**horizontal_size_extension** in sequence_extension()) MPEG-4:14bits
vertical_size_value	StudioVideoObjectLayer()	**video_object_layer_height**	MPEG-2:12bits + 2bits (**vertical_size_extension** in sequence_extension()) MPEG-4:14bits
aspect_ratio_information	StudioVideoObjectLayer()	**aspect_ratio_info**	
frame_rate_code	StudioVideoObjectLayer()	**frame_rate_code**	
bit_rate_value	StudioVideoObjectLayer()	**first_half_bit_rate, latter_half_bit_rate**	MPEG-2:18bits+12bits (**bit_rate_extension** in sequence_extension()) MPEG-4:15bits+15bits
vbv_buffer_size_value	StudioVideoObjectLayer()	**first_half_vbv_buffer_size, latter_half_vbv_buffer_size**	MPEG-2:10bits+8bits (**vbv_buffer_size_extension** in sequence_extension()) MPEG-4:15bits+3bits
constrained_parameters_flag	-	-	used in ISO/IEC 11172-2 (MPEG-1)
load_intra_quantiser_flag	-	**-**	
load_non_intra_quantiser_flag	-	**-**	

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Q.2 Extension and user data

Extension data and user data syntax of MPEG-2 are included in the MPEG-4 Studio Profile.

Q.3 Sequence extension

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
profile_and_level_indication	VisualObjectSequence()	profile_and_level_indication	
progressive_sequence	StudioVideoObjectLayer()	**progressive_sequence**	
chroma_format	StudioVideoObjectLayer()	**chroma_format**	
horizontal_size_extension	-	**-**	See **horizontal_size_value** in sequence header
vertical_size_extension	-	**-**	See **vertical_size_value** in sequence header
bit_rate_extension	-	**-**	See **bit_rate_value** in sequence header
vbv_buffer_size_extension	-	**-**	See **vbv_buffer_size_value** in sequence header
low_delay	StudioVideoObjectLayer()	**low_delay**	
frame_rate_extension_n	-	**-**	
frame_rate_extension_d	-	**-**	

Q.4 Sequence display extension

Sequence display extension of MPEG-2 is included in the MPEG-4 Studio Profile.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Q.5 Group of pictures header

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
time_code	Group_of_StudioVideoObjectPlane()	**time_code_ smpte12m**	Studio Profile uses SMPTE 12M time code MPEG-2:25bits MPEG-4:64bits
closed_gop	Group_of_StudioVideoObjectPlane()	**closed_gov**	
broken_link	Group_of_StudioVideoObjectPlane()	**broken_link**	

Q.6 Picture header

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
temporal_reference	StudioVideoObjectPlane()	**temporal_reference**	
picture_coding_ type	StudioVideoObjectPlane()	**vop_coding_type**	MPEG-2:3bits MPEG-4:2bits
vbv_delay	-	-	not affect decoding process
full_pel_forward_ vector	-	-	used in ISO/IEC 11172-2 (MPEG-1)
forward_f_code	-	-	used in ISO/IEC 11172-2 (MPEG-1)
extra_bit_picture	-	-	not affect decoding process

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Q.7 Picture coding extension

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
f_code[0][0]	StudioVideoObjectPlane()	vop_fcode[0][0]	
f_code[0][1]	StudioVideoObjectPlane()	**vop_fcode[0][1]**	
f_code[1][0]	StudioVideoObjectPlane()	**vop_fcode[1][0]**	
f_code[1][1]	StudioVideoObjectPlane()	**vop_fcode[1][1]**	
intra_dc_precision	StudioVideoObjectPlane()	**intra_dc_precision**	
picture_structure	StudioVideoObjectPlane()	**vop_structure**	
top_field_first	StudioVideoObjectPlane()	**top_field_first**	
frame_pred_frame_dct	StudioVideoObjectPlane()	**frame_pred_frame_dct**	
concealment_motion_vectors	-	**-**	
q_scale_type	StudioVideoObjectPlane()	**q_scale_type**	
intra_vlc_format	-	-	not affect decoding process
alternate_scan	StudioVideoObjectPlane()	**alternate_scan**	
repeat_first_field	StudioVideoObjectPlane()	**repeat_first_field**	
chroma_420_type	-	-	not affect decoding process and exists for historical reasons
progressive_frame	StudioVideoObjectPlane()	**progressive_frame**	
composite_display_flag	-	-	not affect decoding process

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Q.8 Quant matrix extension, Picture display extension, Copy right extension

Quant matrix extension, Picture display extension and Copy right extension of MPEG-2 are included in the MPEG-4 Studio Profile.

Q.9 Slice

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
slice_vertical_ position_extension	-	-	calculated by **macroblock_number**
quantiser_scale_ code	StudioSlice()	**quntiser_scale_ code**	
intra_slice_flag	StudioSlice()	**intra_slice_flag**	
intra_slice	StudioSlice()	**intra_slice**	
reserved_bits	StudioSlice()	**reserved_bits**	
extra_bit_slice	StudioSlice()	**extra_bit_slice**	
extra_information_ slice	StudioSlice()	**extra_information_ slice**	

Q.10 Macroblock

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
macroblock_ escape	-	-	related to **not_coded**
macroblock_ address_increment	-	-	related to **not_coded**
quantiser_scale_ code	StudioMacroblock()	**quantiser_scale_ code**	

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Q.11 Macroblock modes

MPEG-2 4:2:2 Profile) syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
macroblock_type	StudioMacroblock_ modes()	macroblock_type	
frame_motion_type	StudioMacroblock_ modes()	**frame_motion_type**	
field_motion_type	StudioMacroblock_ modes()	**field_motion_type**	
dct_type	StudioMacroblock_ modes()	**dct_type**	

Q.12 Motion vectors

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
motion_vertical_ field_select[0][s]	motion_vectors()	motion_vertical_field_ select[0][s]	
motion_vertical_ field_select[1][s]	motion_vectors()	**motion_vertical_ field_select[1][s]**	

Q.13 Motion vector

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
motion_code[r][s][0]	motion_vector()	motion_code[r][s][0]	
motion_ residual[r][s][0]	motion_vector()	**motion_ residual[r][s][0]**	
dmvvector[0]	-	**-**	
motion_ code[r][s][1]	motion_vector()	**motion_ code[r][s][1]**	
motion_ residual[r][s][1]	motion_vector()	**motion_ residual[r][s][1]**	
dmvvector[1]	-	**-**	

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Q.14 Coded block pattern

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
coded_block_ pattern_420	coded_block_ pattern()	**coded_block_ pattern_420**	Simple Studio Profile does not use 'cbp'
coded_block_ pattern_1	coded_block_ pattern()	**coded_block_ pattern_1**	Simple Studio Profile does not use 'cbp'

Q.15 Block

MPEG-2 4:2:2 Profile syntax	*MPEG-4 Studio Profile*		*Note*
	layer	*syntax*	
dct_dc_size_ luminance	StudioBlock()	**dct_dc_size_ luminance**	MPEG-2:2-9bits MPEG-4:2-14bits
dct_dc_differential	StudioBlock()	**dct_dc_differential**	MPEG-2:1-11bits MPEG-4:1-15bits
dct_dc_size_ chrominance	StudioBlock()	**dct_dc_size_ chrominance**	MPEG-2:2-10bits MPEG-4:2-15bits
Subsequent DCT coefficients	StudioBlock()	Subsequent DCT coefficient	
End of block	StudioBlock()	End of block	MPEG-2:2 or 4bits MPEG-4:3-15 bits

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2004 – All rights reserved

Annex R
(informative)

Forming (RUN, EOP) Symbols

After taking residues between the original VOP and the reconstructed VOP in the base layer, the DCT coefficients in each 8x8 block are obtained (in the case of FGST, the residues are the original VOP). (RUN, EOP) symbols are formed to represent the absolute values of the DCT coefficients. The 64 absolute values of the 8x8 DCT block are zigzag ordered into a one-dimensional array. The formation of (RUN, EOP) symbols is described as the following procedure:

- Find the maximum value of all the absolute values of the DCT coefficients in an fgs vop, and find the minimum number of bits, N, needed to represent this maximum value in the binary format. N is the number of vop-bps for this fgs vop.

- Within each 8x8 block, represent every one of the 64 absolute values of the DCT coefficients with N bits in the binary format and form N block-bps. A block-bp is defined as an array of 64 bits, taken one from each absolute value of the DCT coefficients at the same significant position (after bit shifting if frequency weighting is used).

- Starting from the most significant block-bp (MSB block-bp), 2-D symbols are formed of two components: (a) number of consecutive 0's before a 1 (RUN), (b) whether there are any 1's left on this bit block-bp, i.e. End-Of-Plane (EOP). If a block-bp after the MSB block-bp contains all 0's, a special symbol ALL-ZERO is formed to represent an all-zero block-bp.

The following example illustrates the procedure. Assume that the absolute values after zigzag ordering are given as follows and N = 6:

10, 0, 6, 0, 0, 3, 0, 2, 2, 0, 0, 2, 0, 0, 1, 0, ... 0, 0

The maximum value in this block is found to be 10 and the minimum number of bits to represent 10 in the binary format (1010) is 4. Therefore, two all-zero block-bps before the MSB block-bp are indicated by the fgs_cbp or fgs_msb_not_reached syntax. The remaining 4 block-bps are considered in forming the (RUN, EOP) symbols. Writing every value in the binary format using 4 bits, the 4 block-bps are formed as follows:

1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 (MSB block-bp)

0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0 (Second MSB block-bp)

1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, ... 0, 0 (Third MSB block-bp)

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ... 0, 0 (Fourth MSB block-bp)

Converting the bits of the four block-bps into (RUN, EOP) symbols, we have

(0, 1) (MSB block-bp)

(2, 1) (Second MSB block-bp)

(0, 0), (1,0), (2,0), (1,0), (0, 0), (2, 1) (Third MSB block-bp)

(5, 0), (8, 1) (Fourth MSB block-bp)

Therefore, 10 (RUN, EOP) symbols are formed in this example.

© ISO/IEC 2004 – All rights reserved

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex S
(informative)

A method of decoding a truncated bitstream

In a typical application of FGS, the bitstream at the input of an FGS decoder is a truncated version of the bitstream at the output of an FGS encoder. It is likely that, at the end of each fgs vop bitstream before the next fgs_vop_start_code, only partial bits of the fgs vop are at the input of the decoder due to truncation of the fgs vop bitstream. Decoding of the truncated bitstream is not normative. One possible method for decoding a truncated bitstream of an fgs vop is to look ahead 32 bits at every byte-aligned position in the bitstream indicated by a pointer. If the 32 bits form the fgs_vop_start_code, the decoder either completes decoding up to the fgs_vop_start_code or discards the bits before the fgs_vop_start_code. Otherwise, the first 8 bits of the 32 bits are information bits to be decoded for the fgs vop. The decoder moves forward the pointer by one byte and looks ahead another 32 bits to check for fgs_vop_start_code.

Because not all enhancement layer bits are available for reconstruction of the enhancement DCT coefficients, the decoder may have no enhancement DCT coefficients or partial enhancement DCT coefficients. In such a case, the optimal reconstruction of the DCT coefficients is performed according to the following procedure. Let DIFF be the partial absolute value obtained for a DCT coefficient, SIGN be the sign of the DCT coefficient, and ENH_COEFF be the reconstructed enhancement DCT coefficient.

```
if (DIFF != 0) {
        DIFF += offset
        ENH_COEFF = SIGN*DIFF
}
```

where offset = (decoded_level>2) ? (1<<(decoded_level-3)) : 0 and decoded_level is the index of the last block-bp decoded from the bitstream for an 8x8 DCT block. The decoded_level is ‘1’ for the least significant block-bp and increments its value toward the most significant block-bp in the 8x8 DCT block.

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ICS 35.040

Price based on 706 pages

© ISO/IEC 2004 – All rights reserved
Copyright International Organization for Standardization
Reproduced by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale