INTERNATIONAL STANDARD

ISO/IEC 14496-16

Second edition
2006-12-15

Information technology — Coding of audio-visual objects —

Part 16: Animation Framework eXtension (AFX)

Technologies de l'information — Codage des objets audiovisuels —

Partie 16: Extension du cadre d'animation (AFX)

Reference number
ISO/IEC 14496-16:2006(E)

© ISO/IEC 2006

Copyright International Organization for Standardization
Provided 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 2006

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
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

© ISO/IEC 2006 – All rights reserved

Contents

Page

Foreword v

Introduction vii

1 Scope 1

2 Normative references 1

3 Symbols and abbreviated terms 1

4 Animation Framework eXtension (AFX) 2
4.1 Introduction 2
4.2 The AFX model 3
4.3 Geometry tools 5
4.4 Solid representation 35
4.5 Texture tools 44
4.6 Modeling tools 57
4.7 Generic skeleton, muscle and skin-based model definition and animation 60

5 Bitstream specification 69
5.1 Wavelet Subdivision Surfaces 69
5.2 MeshGrid stream 72
5.3 Depth Image-Based Representation 109
5.4 Compressed Bone-based animation 114
5.5 AFX Generic Backchannel 128
5.6 PointTexture stream 139
5.7 Multiplexing of 3D Compression Streams: the MPEG-4 3D Graphics stream (.m3d) syntax 149

6 AFX Object code 152

7 3D Graphics Profiles 153
7.1 Introduction 153
7.2 "Graphics" Dimension 153
7.3 "Scene Graph" Dimension 155
7.4 "3D Compression" Dimension 158

8 XMT representation for AFX tools 161
8.1 AFX nodes 161
8.2 AFX encoding hints 161
8.3 AFX encoding parameters 162
8.4 AFX decoder specific info 165
8.5 XMT for Bone-based Animation 166

Annex A (normative) **Wavelet Mesh Decoding Process** 171
A.1 Overview 171
A.2 Base mesh 171
A.3 Definitions and notations 171
A.4 Bitplanes extraction 172
A.5 Zero-tree decoder 173
A.6 Synthesis filters and mesh reconstruction 173
A.7 Basis change 174

Annex B (normative) MeshGrid Representation 176
B.1 The hierarchical multi-resolution MeshGrid 176
B.2 Scalability Modes 180
B.3 Animation Possibilities 183

© ISO/IEC 2006 – All rights reserved

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

Annex C (informative) **MeshGrid representation** **186**
C.1 Representing Quadrilateral meshes in the MeshGrid format **186**
C.2 IndexedFaceSet models represented by the MeshGrid format. **188**
C.3 Computation of the number of ROIs at the highest resolution level given an optimal ROI size **189**

Annex D (informative) **Solid representation** **190**
D.1 Overview **190**
D.2 Solid Primitives **190**
D.3 Arithmetics of Forms **191**

Annex E (informative) **Face and Body animation: XMT compliant animation and encoding parameter file format** **197**
E.1 XSD for FAP file format **197**
E.2 XSD for BAP file format **198**
E.3 XSD for EPF **199**

Annex F (normative) **Node coding tables** **203**
F.1 Node Coding tables **203**
F.2 Node Definition Type tables **203**

Annex G (Informative) **Patent statements** **204**

Bibliography **205**

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided 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.

ISO/IEC 14496-16 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 14496-16:2004), which has been technically revised. It also incorporates the amendment ISO/IEC 14496-16:2004/Amd.1:2006 and the Technical Corrigenda ISO/IEC 14496-16:2004/Cor.1:2005 and ISO/IEC 14496-16:2004/Cor.2:2005.

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* [Technical Report]
- *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*
- *Part 9: Reference hardware description* [Technical Report]
- *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) extensions*
- *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*

© ISO/IEC 2006 – All rights reserved

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

— *Part 19: Synthesized texture stream*

— *Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)*

— *Part 21: MPEG-J GFX*

— *Part 22: Open Font Format*

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

The 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 statement of the holder of this patent right is registered with the ISO and IEC. Information may be obtained from the companies listed in Annex G.

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

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided 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 16:
Animation Framework eXtension (AFX)

1 Scope

This International Standard specifies MPEG-4 Animation Framework eXtension (AFX) model for creating interactive multimedia contents by composing natural and synthetic objects. Within this model, MPEG-4 is extended with higher-level synthetic objects for geometry, texture, and animation as well as dedicated compressed representations.

AFX also specifies a backchannel for progressive streaming of view-dependent information.

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 14496-1:2004, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-2:2004, *Information technology — Coding of audio-visual objects — Part 2: Visual*

ISO/IEC 14496-11:2005, *Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*

3 Symbols and abbreviated terms

List of symbols and abbreviated terms.

AFX	Animation Framework eXtension
BIFS	BInary Format for Scene
DIBR	Depth-Image Based Representation
ES	Elementary Stream
IBR	Image-Based Rendering
NDT	Node Data Type
OD	Object Descriptor
VRML	Virtual Reality Modelling Language

© ISO/IEC 2006 – All rights reserved

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

4 Animation Framework eXtension (AFX)

4.1 Introduction

"Most people think the word 'animation' means movement.
But it doesn't. It comes from 'animus' which means 'life or to live'.
Making it move is not animation, but just the mechanics of it."
Frank Thomas and Ollie Johnston
Disney Animation: the illusion of life, 1981

MPEG-4, ISO/IEC 14496, is a multimedia standard that enables composition of multiple audio-visual objects on a terminal. Audio and visual objects can come from *natural* sources (e.g. a microphone, a camera) or from *synthetic* ones (i.e. made by a computer); each source is called a *media* or a *stream*. On their terminals, users can display, play, and interact with MPEG-4 audio-visual contents, which can be downloaded previously or streamed from remote servers. Moreover, each object may be protected to ensure a user has the right credentials before downloading and displaying it.

Unlike natural audio and video objects, computer graphics objects are purely synthetic. Mixing computer graphics objects with traditional audio and video enables augmented reality applications, i.e. applications mixing natural and synthetic objects. Examples of such contents range from DVD menus, and TV's Electronic Programming Guides to medical and training applications, games, and so on.

Like other computer graphics specifications, MPEG-4 synthetic objects are organized in a *scene graph* based on VRML97 [1], which is a direct acyclic tree where *nodes* represent objects and branches their properties, called *fields*. As each object can receive and emit events, two branches can be connected by the means of a *route*, which propagates events from one field of one node to another field of another node. As any other MPEG-4 media, scenes may receive updates from a server that modify the topology of the scene graph.

The Animation Framework eXtension (AFX) proposes a conceptual organization of synthetic models for computer animations as well as specific compression schemes; models defined in ISO/IEC 14496-11 and in this part fit in organization (see Figure 2).

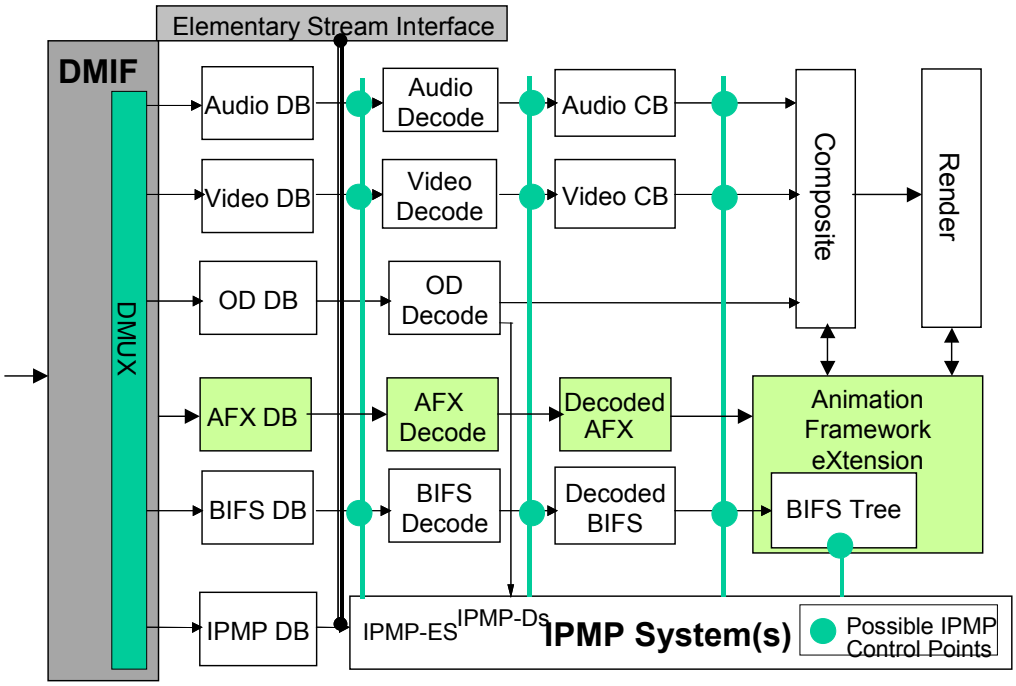


Figure 1 — Animation Framework and MPEG-4 Systems.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Figure 1 shows the position of the Animation Framework within MPEG-4 Terminal architecture. It extends the existing BIFS tree with new tools and define the AFX streams that carry dedicated compressed object representations and a backchannel for view-dependent features.

4.2 The AFX model

To understand the AFX model, let's take an example. Suppose one wants to build an avatar. The avatar consists of geometry elements that describe its legs, arms, head and so on. Simple geometric elements can be used and deformed to produce more physically realistic geometry. Then, skin, hair, cloths are added. These may be physic-based models attached to the geometry. Whenever the geometry is deformed, these models deform and thanks to their physics, they may produce wrinkles. Biomechanical models are used for motion, collision response, and so on. Finally, our avatar may exhibit special behaviors when it encounters objects in its world. It might also learn from experiences: for example, if it touches a hot surface and gets hurt, next time, it will avoid touching it.

This hierarchy also works in a top to bottom manner: if it touches a hot surface, its behavior may be to retract its hand. Retracting its hand follows a biomechanical pattern. The speed of the movement is based on the physical property of its hand linked to the rest of its body, which in turn modify geometric properties that define the hand.

AFX defines 6 groups of components, following [38].

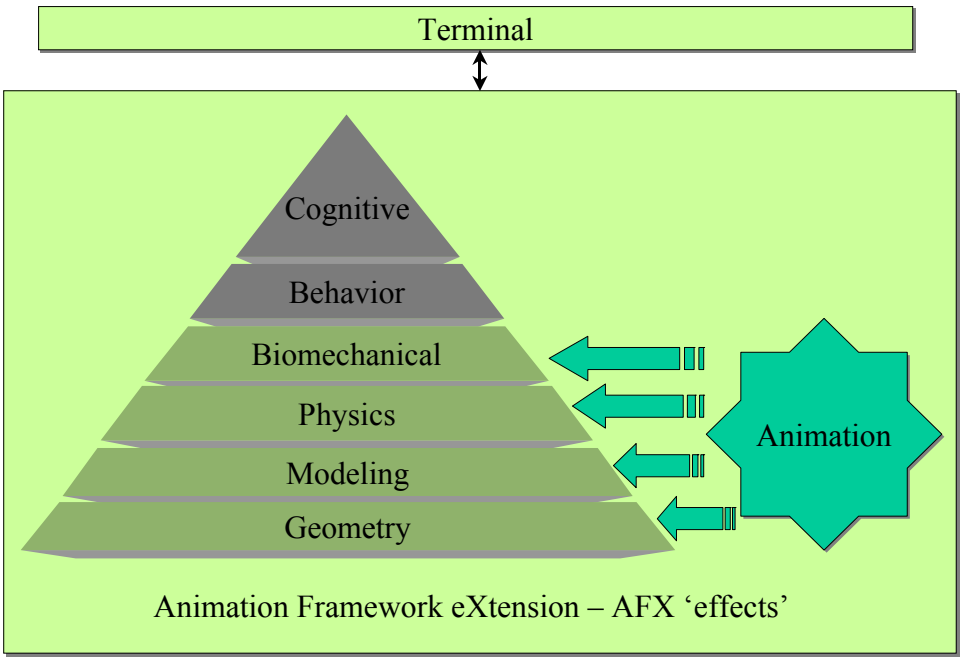


Figure 2 — Models in computer games and animation [38].

a) *Geometric component*. These models capture the form and appearance of an object. Many characters in animations and games can be quite efficiently controlled at this low-level. Due to the predictable nature of motion, building higher-level models for characters that are controlled at the geometric level is generally much simpler.

b) *Modeling component*. These models are an extension of geometric models and add linear and non-linear deformations to them. They capture transformation of the models without changing its original shape. Animations can be made on changing the deformation parameters independently of the geometric models.

c) *Physical component*. These models capture additional aspects of the world such as an object's mass inertia, and how it responds to forces such as gravity. The use of physical models allows many motions to be created automatically and with unparallel realism.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

d) *Biomechanical component*. Real animals have muscles that they use to exert forces and torques on their own bodies. If we already have built physical models of characters, they can use virtual muscles to move themselves around. These models have their roots in control theory.

e) *Behavioral component*. A character may expose a *reactive behavior* when its behavior is solely based on its perception of the current situation (i.e. no memory of previous situations). *Goal-directed behaviors* can be used to define a cognitive character's goals. They can also be used to model *flocking behaviors*.

f) *Cognitive component*. If the character is able to learn from stimuli from the world, it may be able to adapt its behavior. These models are related to artificial intelligence techniques.

AFX specification currently deals with the first four categories. Models of the last two categories are typically application-specific and often designed programmatically. Similarly, while the first four categories can be animated using existing tools such as interpolators, the last two categories have their own logic and cannot be animated the same way.

In each category, one can find many models for any applications: from simple models that require little processing power (low-level models) to more complex models (high-level models) that require more computations by the terminal. VRML [1] and BIFS specifications provide low-level models that belong to the Geometry component with the exception of Face and Body Animation tools that belong to the Biomechanical component.

Higher-level components can be defined as providing a compact representation of functionality in a more abstract manner. Typically, this abstraction leads to mathematical models that need few parameters. These models cannot be rendered directly by a graphic card: internally, they are converted to low-level primitives a graphic card can render. Besides a more compact representation, this abstraction often provides other functionalities such as but not limited to compact representation, view-dependent subdivision, automatic level-of-details, smoother graphical representation , scalability across terminals, and progressive local refinements.

For example, a subdivision surface can be subdivided based on the area viewed by the user. For animations, piecewise-linear interpolators require few computations but require lots of data in order to represent a curved path. Higher-level animation models represent animation using piecewise-spline interpolators with less values and provide more control over the animation path and timeline.

In the remaining of this document, this conceptual organization of tools is followed in the same spirit an author will create content: the geometry is first defined with or without solid modeling tools, then texture is added to it. Objects can be deformed and animated using modeling and animation tools. Finally, avatars need biomechanical tools. Behavioral and Cognitive models can be programmatically implemented using JavaScript or MPEG-J defined in ISO/IEC 14496-11.

NOTE

Some generic tools developed originally within the Animation Framework eXtension have been relocated in ISO/IEC 14496-11/AMD1 along with other generic tools. This includes:

— Spline-based generic animation tools, called Animator nodes;

— Optimized interpolator compression tools;

— BitWrapper node that enables compressed representation of exisiting nodes;

— Procedural textures based on fractal plasma.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.3 Geometry tools

4.3.1 Introduction

Geometry tools consist of the following technologies:

— NURBS, which consists of the following nodes: **NurbsSurface**, **NurbsCurve**, and **NurbsCurve2D**;

— Subdivision surfaces consisting of the following nodes: **SubdivisionSurface**, **SubdivSurfaceSector** and **WaveletSubdivisionSurface** of which the latter enables to add wavelet-encoded details at different resolutions to subdivision surfaces;

— MeshGrid, which consists of the **MeshGrid** node;

4.3.2 Non-Uniform Rational B-Spline (NURBS)

4.3.2.1 Introduction

A Non-Uniform Rational B-Spline (NURBS) curve of degree $p > 0$ (and hence of order $p+1$) and control points $\{\mathbf{P}_i\}$ ($0 \le i \le n-1$; $n \ge p+1$) has for Equation 1 [37], [63], [88]:

$$\mathbf{C}(u) = \sum_{i=0}^{n-1} R_{i,p}(u)\mathbf{P}_i = \frac{\sum_{i=0}^{n-1} N_{i,p}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n-1} N_{i,p}(u) w_i}$$

Equation 1 — NURBS curve definition.

The parameter $u \in [0, 1]$ allows to travel along the curve and $\{R_{i,p}\}$ are its rational basis functions. The latter can in turn be expressed in terms of some positive (and not all null) weights $\{w_i\}$, and $\{N_{i,p}\}$, the pth-degree B-Spline basis functions defined on the possibly non-uniform, but always non-decreasing knot sequence/vector of length $m = n + p+1$: $U = \{u_0, u_1, \ldots, u_{m-1}\}$, where $0 \le u_i \le u_{i+1} \le 1 \ \forall\ 0 \le i \le m - 2$.

The B-Spline basis functions are defined recursively using the Cox de Boor formula:

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1 & \text{if } u_i \le u < u_{i+1}; \\ 0 & \text{otherwise;} \end{cases} \\ N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u). \end{aligned}$$

If $u_i = u_{i+1} = \ldots = u_{i+k-1}$, it is said that u_i has multiplicity k. $\mathbf{C}(u)$ is infinitely differentiable inside knot spans, *i.e.*, for $u_i < u < u_{i+1}$, but only $p–k$ times differentiable at the parameter value corresponding to a knot of multiplicity k, so setting several consecutive knots to the same value u_j decreases the smoothness of the curve at u_j. In general, knots cannot have multiplicity greater than p, but the first and/or last knot of U can have multiplicity $p+1$, *i.e.*, $u_0 = \ldots = u_p = 0$ and/or $u_{m-p-1} = \ldots = u_{m-1} = 1$, which causes $\mathbf{C}$ to interpolate the corresponding endpoint(s) of the control polygon defined by $\{\mathbf{P}_i\}$, *i.e.*, $\mathbf{C}(0) = \mathbf{P}_0$ and/or $\mathbf{C}(1) = \mathbf{P}_{n-1}$. Therefore, a knot vector of the kind $U = \left\{ \underbrace{u_0 = 0, \ldots, 0}_{p+1}, u_{p+1}, \ldots, u_{m-p-2}, \underbrace{1, \ldots, 1 = u_{m-1}}_{p+1} \right\}$ causes the curve to be endpoint interpolating, *i.e.*, to interpolate both endpoints of its control polygon. Extreme knots, multiple or not, may enclose any non-decreasing subsequence of interior knots: $0 < u_i \le u_{i+1} < 1$. An endpoint interpolating curve with no interior knots, *i.e.*, one with U consisting of $p+1$ zeroes followed by $p+1$ ones, with no other values in between, is a pth-

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

degree Bézier curve: *e.g.*, a cubic Bézier curve can be described with four control points (of which the first and last will lie on the curve) and a knot vector $U = \{0, 0, 0, 0, 1, 1, 1, 1\}$.

It is possible to represent all types of curves with NURBS and, in particular, all conic curves (including parabolas, hyperbolas, ellipses, etc.) can be represented using rational functions, unlike when using merely polynomial functions.

Other interesting properties of NURBS curves are the following:

— Affine invariance: rotations, translations, scalings, and shears can be applied to the curve by applying them to $\{\mathbf{P}_i\}$.

— Convex hull property: the curve lies within the convex hull defined by $\{\mathbf{P}_i\}$. The control polygon defined by $\{\mathbf{P}_i\}$ represents a piecewise approximation to the curve. As a general rule, the lower the degree, the closer the NURBS curve follows its control polygon.

— Local control: if the control point $\mathbf{P}_i$ is moved or the weight w_i is changed, only the portion of the curve swept when $u_i < u < u_{i+p+1}$ is affected by the change.

NURBS surfaces are defined as tensor products of two NURBS curves, of possibly different degrees and/or numbers of control points:

$$\mathbf{C}(u,v) = \frac{\sum_{i=0}^{n-1}\sum_{j=0}^{l-1} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n-1}\sum_{j=0}^{l-1} N_{i,p}(u)N_{j,q}(v)w_{i,j}}$$

Equation 2 — NURBS surface definition.

The two independent parameters $u, v \in [0, 1]$ allow to travel across the surface. The B-Spline basis functions are defined as previously, and the resulting surface has the same interesting properties that NURBS curves have. Multiplicity of knots may be used to introduce sharp features (corners, creases, etc.) in an otherwise smooth surface, or to have it interpolate the perimeter of its control polyhedron.

4.3.2.2 NurbsCurve

4.3.2.2.1 Node interface

```
NurbsCurve { #%NDT=SFGeometryNode
    eventIn       MFInt32      set_colorIndex
    exposedField  SFColorNode  color          NULL
    exposedField  MFVec4f      controlPoint    []
    exposedField  SFInt32      tessellation    0    # [0, ∞)
    field         MFInt32      colorIndex      [] # [-1, ∞)
    field         SFBool       colorPerVertex  TRUE
    field         MFFloat      knot            []   # (-∞, ∞)
    field         SFInt32      order           4    # [3, 34]
}
```

4.3.2.2.2 Functionality and semantics

The **NurbsCurve** node describes a 3D NURBS curve, which is displayed as a curved line, similarly to what is done with the **IndexedLineSet** primitive.

The **order** field defines the order of the NURBS curve, which is its degree plus one.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **controlPoint** field defines a set of control points in a coordinate system where the weight is the last component. The number of control points must be greater than or equal to the order of the curve. All weight values must be greater than or equal to 0, and at least one weight must be strictly greater than 0. If the weight of a control point is increased above 1, that point is more closely approximated by the surface. However the surface is not changed if all weights are multiplied by a common factor.

The **knot** field defines the knot vector. The number of knots must be equal to the number of control points plus the order of the curve, and they must be ordered non-decreasingly. By setting consecutive knots to the same value, the degree of continuity of the curve at that parameter value is decreased. If *o* is the value of the field **order**, *o* consecutive knots with the same value at the beginning (resp. end) of the knot vector cause the curve to interpolate the first (resp. last) control point. Other than at its extremes, there may not be more than $o-1$ consecutive knots of equal value within the knot vector. If the length of the knot vector is 0, a default knot vector consisting of *o* 0's followed by *o* 1's, with no other values in between, will be used, and a Bézier curve of degree $o-1$ will be obtained. A closed curve may be specified by repeating the starting control point at the end and specifying a periodic knot vector.

The **tessellation** field gives hints to the curve tessellator as to the number of subdivision steps that must be used to approximate the curve with linear segments: for instance, if the value *t* of this field is greater than or equal to that of the **order** field, *t* can be interpreted as the absolute number of tesselation steps, whereas $t = 0$ lets the browser choose a suitable tessellation.

Fields **color**, **colorIndex**, **colorPerVertex**, and **set_colorIndex** have the same semantic as for **IndexedLineSet** applied to the control points.

4.3.2.3 NurbsCurve2D

4.3.2.3.1 Node interface

```
NurbsCurve2D { #%NDT=SFGeometryNode
    eventIn       MFInt32        set_colorIndex
    exposedField  SFColorNode    color          NULL
    exposedField  MFVec3f        controlPoint   []
    exposedField  SFInt32        tessellation   0    # [0, ∞)
    field         MFInt32        colorIndex     [] # [-1, ∞)
    field         SFBool         colorPerVertex TRUE
    field         MFFloat        knot           []   # (-∞, ∞)
    field         SFInt32        order          4    # [3, 34]
}
```

4.3.2.3.2 Functionality and semantics

The **NurbsCurve2D** is the 2D version of **NurbsCurve**; it follows the same semantic as **NurbsCurve** with 2D control points.

4.3.2.4 NurbsSurface

4.3.2.4.1 Node interface

```
NurbsSurface { #%NDT=SFGeometryNode
    eventIn       MFInt32                set_colorIndex
    eventIn       MFInt32                set_texCoordIndex
    exposedField  SFColorNode            color          NULL
    exposedField  MFVec4f                controlPoint   []
    exposedField  SFTextureCoordinateNode texCoord      NULL
    exposedField  SFInt32                uTessellation  0    # [0, ∞)
    exposedField  SFInt32                vTessellation  0    # [0 ,∞)
    field         MFInt32                colorIndex     [] # [-1, ∞)
```

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    field       SFBool      colorPerVertex  TRUE
    field       SFBool      solid           TRUE
    field       MFInt32     texCoordIndex   []   # [-1 ,∞)
    field       SFInt32     uDimension      4    # [3, 258]
    field       MFFloat     uKnot           []   # (-∞,∞)
    field       SFInt32     uOrder          4    # [3, 34]
    field       SFInt32     vDimension      4    # [3, 258]
    field       MFFloat     vKnot           []   # (-∞,∞)
    field       SFInt32     vOrder          4    # [3, 34]
}
```

4.3.2.4.2 Functionality and semantics

The **NurbsSurface** Node describes a 3D NURBS surface. Similar 3D surface nodes are **ElevationGrid** and **IndexedFaceSet**. In particular, **NurbsSurface** extends the definition of **IndexedFaceSet**. If an implementation does not support **NurbsSurface**, it should still be able to display its control polygon as a set of (triangulated) quadrilaterals, which is the coarsest approximation of the NURBS surface.

The **uOrder** field defines the order of the surface in the *u* dimension, which is its degree in the *u* dimension plus one. Similarly, the **vOrder** field define the order of the surface in the *v* dimension.

The **uDimension** and **vDimension** fields define respectively the number of control points in the *u* and *v* dimensions, which must be greater than or equal to the respective orders of the curve.

The **controlPoint** field defines a set of control points in a coordinate system where the weight is the last component. These control points form an array of dimension **uDimension** x **vDimension**, in a similar way to the regular grid formed by the control points of an **ElevationGrid**, the difference being that, in the case of a **NurbsSurface**, the points need not be regularly spaced. The number of control points in each dimension must be greater than or equal to the corresponding order of the curve. Specifically, the three spatial coordinates (*x*, *y*, *z*) and weight (*w*) of control point $\mathbf{P}_{i,j}$ (NB: $0 \le i < \mathbf{uDimension} \ge \mathbf{uOrder}$ and $0 \le j < \mathbf{vDimension} \ge \mathbf{vOrder}$) are obtained as follows:

```
P[i,j].x = controlPoint[i + j*uDimension].x

P[i,j].y = controlPoint[i + j*uDimension].y

P[i,j].z = controlPoint[i + j*uDimension].z

P[i,j].w = controlPoint[i + j*uDimension].w
```

All **weight** values must be greater than or equal to 0, and at least one weight must be strictly greater than 0. If the weight of a control point is increased above 1, that point is more closely approximated by the surface. However the surface is not changed if all weights are multiplied by a common factor.

The **uKnot** and **vKnot** fields define the knot vectors in the *u* and *v* dimensions respectively, and their semantics are analogous to that of the **knot** field of the **NurbsCurve** node.

The **uTessellation** and **vTessellation** fields give hints to the surface tessellator as to the number of subdivision steps that must be used to approximate the surface with (triangulated) quadrilaterals: for instance, if the value *t* of the **uTessellation** field is greater than or equal to that of the **uOrder** field, *t* can be interpreted as the absolute number of tesselation steps in the *u* dimension, whereas $t = 0$ lets the browser choose a suitable tessellation.

solid is a rendering hints that indicates if the surface is closed. If FALSE, two-side lighting is enabled. If TRUE, only one-side lighting is applied.

color, **colorIndex, colorPerVertex, set_colorIndex, texCoord**, **texCoordIndex,** and **set_textCoordIndex** have the same functionality and semantics as for **IndexedFaceSet** and they apply

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

on the control points defined in **controlPoint**, just like these fields apply on **coord** points of an **IndexedFaceSet**.

4.3.3 Subdivision surfaces

4.3.3.1 Introduction

Subdivision is a recursive refinement process that splits the facets or vertices of a polygonal mesh (the initial "control hull") to yield a smooth limit surface. The refined mesh obtained after each subdivision step is used as the control hull for the next step, so all successive (and hierarchically nested) meshes can be regarded as control hulls. The refinement of a mesh is performed both on its topology, as the vertex connectivity is made richer and richer, and on its geometry, as the new vertices are positioned in such a way that the angles formed by the new facets are smaller than those formed by the old facets.

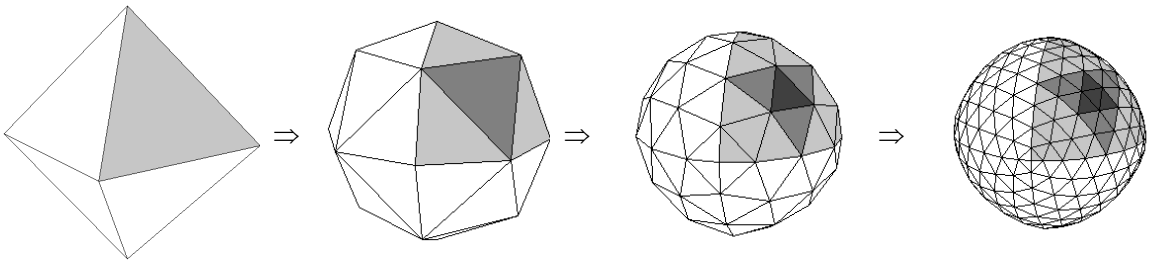


Figure 3 — Nested control meshes.

Figure 3 shows an example of how subdivision would be applied to a triangular mesh. At each step, one triangle is shaded to highlight the correspondence between one level and the next: each triangle is broken down into four new triangles, whose vertex positions are perturbed to have a smoother and smoother surface.

4.3.3.1.1 Piecewise Smooth Surfaces

Subdivision schemes for piecewise smooth surfaces include common modeling primitives such as quadrilateral free-form patches with creases and corners.

A somewhat informal description of piecewise smooth surfaces is given here; for simplicity, only surfaces without self intersections are considered. For a closed C^1-continuous surface in $\Re^3$, each point has a neighborhood that can be smoothly deformed into an open planar disk *D*. A surface with a *smooth boundary* can be described in a similar way, but neighborhoods of boundary points can be smoothly deformed into a half-disk *H*, with closed boundary (Figure 4). In order to allow *piecewise* smooth boundaries, two additional types of local charts are introduced: concave and convex corner charts, Q_1 and Q_3.

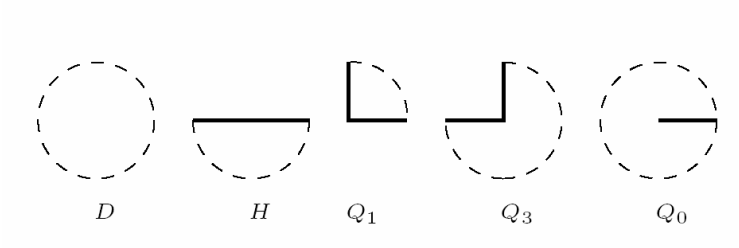


Figure 4 — The charts for a surface with piecewise smooth boundary.

Piecewise-smooth surfaces are constructed out of surfaces with piecewise smooth boundaries joined together. If two surface patches have a common boundary, but different normal directions along the boundary, the resulting surface has a sharp crease.

© ISO/IEC 2006 – All rights reserved

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

Two adjacent smooth segments of a boundary are allowed to be joined, producing a crease ending in a *dart* (cf. [40]). For dart vertices an additional chart Q_{-1} is required; the surface near a dart can be deformed into this chart smoothly everywhere except at an open edge starting at the center of the disk.

4.3.3.1.2 Tagged meshes

Before describing the set of subdivision rules, the description of the tagged meshes is described, which the algorithms accept as input. These meshes represent piecewise-smooth surfaces with features described above.

The complete list of tags is as follows. Edges can be tagged as *crease edges*. A vertex with incident crease edges receives one of the following tags:

— *Crease vertex:* joins exactly two incident crease edges smoothly.

— *Corner vertex:* connects two or more creases in a corner (convex or concave).

— *Dart vertex:* causes the crease to blend smoothly into the surface; this is the only allowed type of crease termination at an interior non-corner vertex.

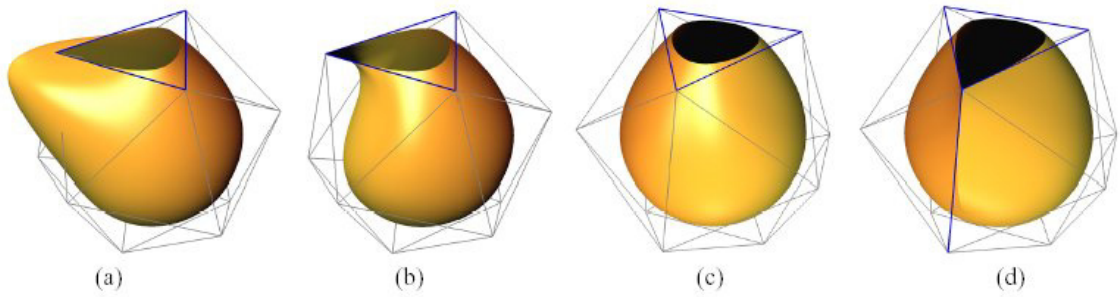


Figure 5 — Features: (a) concave corner, (b) convex corner, (c) smooth boundary/crease, (d) corner with two convex sectors.

Boundary edges are automatically assigned crease tags; boundary vertices that do not have a corner tag are assumed to be smooth crease vertices.

Crease edges divide the mesh into separate patches, several of which can meet in a corner vertex. At a corner vertex, the creases meeting at that vertex separate the ring of triangles around the vertex into sectors. Each sector of the mesh is labeled as *convex sector* or *concave sector* indicating how the surface should approach the corner.

In all formulas k denotes the number of faces incident at a vertex in a given sector. Note that this is different from the standard definition of the vertex degree: faces, rather than edges are counted. Both quantities coincide for interior vertices with no incident crease edges. This number is referred as *crease degree*.

The only restriction on sector tags is that concave sectors consist of at least two faces. An example of a tagged mesh is given in Figure 6.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

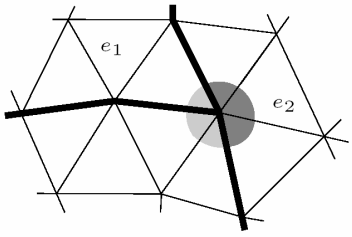


Figure 6 — Crease edges meeting in a corner with two convex (light grey) and one concave (dark grey) sectors. The subdivision scheme modifies the rules for edges incident to crease vertices (e.g. e_1) and corners (e.g. e_2).

4.3.3.2 SubdivisionSurface

4.3.3.2.1 Node interface

```
SubdivisionSurface { #%NDT=SFGeometryNode, SFBaseMeshNode
    eventIn          MFInt32     set_colorIndex
    eventIn          MFInt32     set_coordIndex
    eventIn          MFInt32     set_texCoordIndex
    eventIn          MFInt32     set_creaseEdgeIndex
    eventIn          MFInt32     set_cornerVertexIndex
    eventIn          MFInt32     set_creaseVertexIndex
    eventIn          MFInt32     set_dartVertexIndex
    exposedField     SFColorNode                color              NULL
    exposedField     SFCoordinateNode           coord              NULL
    exposedField     SFTextureCoordinateNode    texCoord           NULL
    exposedField     SFInt32     subdivisionType      0        # 0 – midpoint, 1 – Loop, 2 – butterfly,
                                                              # 3 – Catmull-Clark
    exposedField     SFInt32     subdivisionSubType   0        # 0 – Loop, 1 – Warren-Loop
    field            MFInt32     invisibleEdgeIndex   []       # [0, ∞)
                                                              # extended Loop if non-empty
    exposedField     SFInt32     subdivisionLevel     0        # [-1, ∞)
    # IndexedFaceSet fields
    field            SFBool      ccw                  TRUE
    field            MFInt32     colorIndex           []       # [-1, ∞)
    field            SFBool      colorPerVertex       TRUE
    field            SFBool      convex               TRUE
    field            MFInt32     coordIndex           []       # [-1, ∞)
    field            SFBool      solid                TRUE
    field            MFInt32     texCoordIndex        []       # [-1, ∞)
    # tags
    field            MFInt32     creaseEdgeIndex      []       # [-1, ∞)
    field            MFInt32     dartVertexIndex      []       # [-1, ∞)
    field            MFInt32     creaseVertexIndex    []       # [-1, ∞)
    field            MFInt32     cornerVertexIndex    []       # [-1, ∞)
    # sector information
    exposedField     MFSubdivSurfaceSectorNode sectors              []
}
```

4.3.3.2.2 Functionality and semantics

The **SubdivisionSurface** node is similar to the existing **IndexedFaceSet** node with all fields relating to face normals and the **creaseAngle** field removed since normals are generated by the subdivision algorithm at run-time and their behavior at facet boundaries is controlled by edge and vertex crease tagging. The control hull is specified as a collection of polygons and must be a manifold mesh. No “dangling” edges or vertices are

© ISO/IEC 2006 – All rights reserved

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

permitted, and each polygon may share each of its edges with at most one other polygon. More details about the subdivision process and rules are described in subclause 4.3.3.2.3.

The **SubdivisionSurface** node represents a 3D shape formed through surface subdivision of a control hull constructed from vertices listed in the **coord** field. The **coord** field contains a **Coordinate** node that defines the 3D vertices referenced by the **coordIndex** field. **SubdivisionSurface** uses the indices in its **coordIndex** field to specify the polygonal faces of the control hull by indexing into the coordinates in the **Coordinate** node. An index of "-1" is used as a separator for the control hull faces. The last face may be (but does not have to be) followed by a "-1" index. If the greatest index in the **coordIndex** field is N, then the **Coordinate** node shall contain N+1 coordinates (indexed as 0 to N). Each face of the **SubdivisionSurface** control hull shall have:

— exactly three non-coincident vertices for triangular schemes (midpoint, Loop and butterfly);

— at least three non-coincident vertices for quadrilateral schemes (Catmull-Clark and extended Loop);

— vertices that define a planar, non-self-intersecting polygon.

Otherwise, the results are undefined.

The **SubdivisionSurface** node is specified in the local coordinate system and is affected by the transformation of its ancestors.

The **subdivisionType** field specifies the major subdivision scheme:

0. Midpoint (non-smoothing);
1. Loop (subclause 4.3.3.2.3.1);
2. Dyn's butterfly (cf. [34], [94]);
3. Catmull-Clark (subclause 4.3.3.2.3.3).

The **subdivisionSubType** field specifies a sub category of subdivision rules in the major subdivision rule. For this specification, for **subdivisionType** 0 (Loop), **subdivisionSubType** 0 indicates Loop and 1 Warren's Loop.

A **subdivisionType** value of zero and a non-empty **invisibleEdgeIndex** array imply that the extended Loop scheme (subclause 4.3.3.2.3.2) is used.

The **invisibleEdgeIndex** field is used to select co-ordinate pairs from **coord** to define which edges should be tagged as invisible. Consecutive pairs of co-ordinates define an edge; this pair of vertices must belong to one facet as defined by the **coordIndex** field and may be "interior" to the facet (e.g. in the case of a quadrilateral, the invisible edge would be one of its diagonals). All non-triangular facets should have enough tags to fully describe their triangulation. Note that a non-empty **invisibleEdgeIndex** array is only allowed when **subdivisionType** is 0; the results are undefined in other cases.

The **subdivisionLevel** field specifies the number of times the surface should be subdivided. If this field is "-1" then the terminal should apply enough iterations of subdivision until it reaches a "visually smooth" surface.

The fields **color**, **coord**, **texCoord**, **ccw**, **colorIndex**, **colorPerVertex**, **convex**, **coordIndex**, **solid**, and **texCoordIndex** have the same semantic as for the **IndexedFaceSet** node. These fields define the control hull of the subdivision surface. Textures coordinates and color information, if present, propagate from the control hull to subsequent subdivision levels through linear interpolation.

The **creaseEdgeIndex** field containsan indexed line set of creases, indicating where creases appear on the otherwise smooth subdivision surface (subclause 4.3.3.1.2). As for the **IndexedLineSet** node, the line is defined by successive vertex indices defined in the **coord** field. An index of "-1" indicates that the current

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

crease has ended and the next one begins. The last crease may be (but does not have to be) followed by a "-1". Crease edges must already exist in the control hull topology (i.e. the edge must coincide with a single edge of a polygon as defined by the **coordIndex** field). Note that if an edge is tagged as both invisible and a crease, the results are undefined.

From the **creaseEdgeIndex** array, the default behavior (and appropriate subdivision rules) for vertices on crease edges are defined (subclause 4.3.3.1.2) as:

- Vertices with exactly one crease edge in their neighborhood are dart vertices,
- Vertices with exactly two crease edges in their neighborhood are crease vertices,
- Vertices with three or more crease edges in their neighborhood are corner vertices.

However, this array may not provide a description general enough for all surfaces and three arrays are provided: **cornerVertexIndex**, **creaseVertexIndex**, and **dartVertexIndex**. The information provided by these arrays completes (or overrides) the information in **creaseEdgeIndex**.

The **cornerVertexIndex** field indicates the vertex index of corner vertices.

The **creaseVertexIndex** field indicates the vertex index of crease vertices.

The **dartVertexIndex** field indicates the vertex index of dart vertices.

Finally, the **sectors** field may optionally contain an array of **SubdivSurfaceSector** nodes that describe additional sector information; see subclause 4.3.3.2.3 for more details. Note that if both the **sector** and the **invisibleEdgeIndex** fields are non-empty, the results are undefined.

4.3.3.2.3 Subdivision Stencils

In order to reduce the angles between adjacent triangles, the position of each new vertex introduced by the subdivision process is calculated as a function of some old vertices in the vicinity of the considered edge. The position of old vertices themselves may be altered as well for that same purpose.

The *stencil* of the subdivision scheme refers to the set of old vertices used to compute a new vertex. The term *mask,* or the set of (averaging) rules, is used to refer to the set of values of coefficients. It determines the convergence of the process and the smoothness degree of the limit surface.

Different masks are necessary for different types of vertices discussed in more detail in the next subclause.

4.3.3.2.3.1 Loop subdivision

Figure 7 shows the stencils used for repositioning old vertices and inserting new ones by splitting edges in the case of Loop's scheme [51].

Vertex rules are used to update positions of old vertices, edge rules are used to compute positions of newly inserted vertices.

For untagged interior and dart vertices rules shown in Figure 7a are used; for boundary/crease vertices rules in Figure 7b are used. Positions of corner vertices are never changed.

© ISO/IEC 2006 – All rights reserved

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

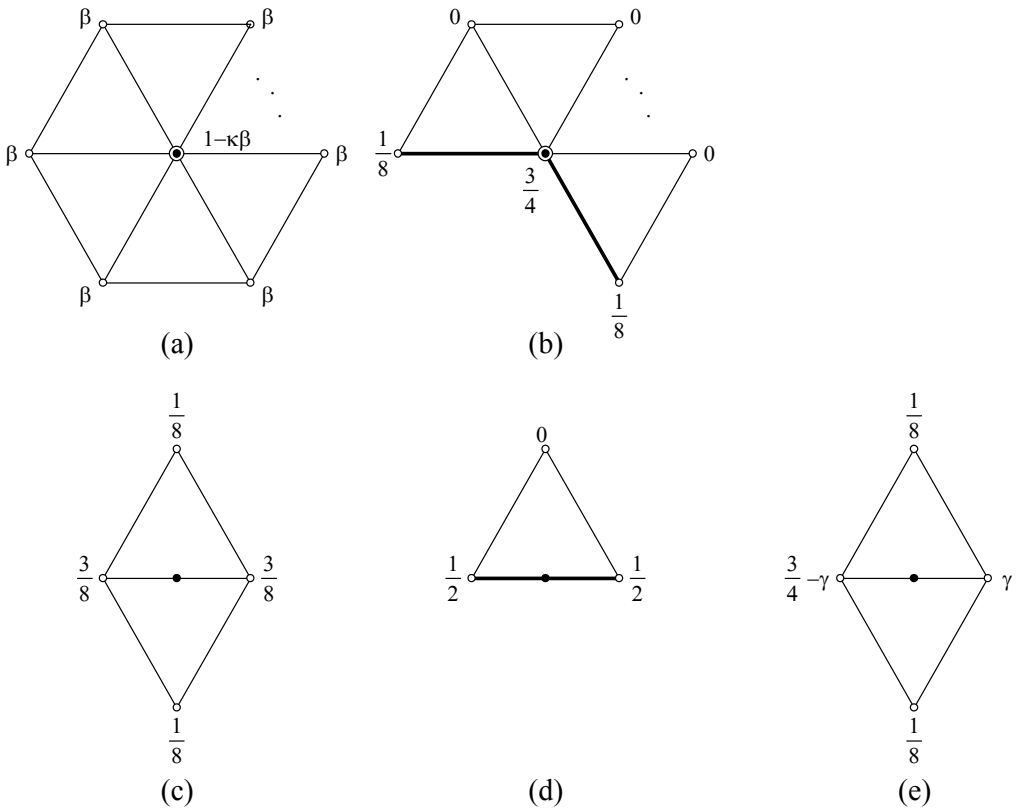


Figure 7 — Stencils of Loop's subdivision scheme. (a) repositioning interior old vertices (b) repositioning boundary or crease old vertices , (c) splitting interior edges not adjacent to boundary extraordinary vertices, (d) splitting boundary and crease edges, (e) splitting interior edges adjacent to an extraordinary (valence not 4) vertex on the boundary/crease or corner vertex . The values of beta and gamma are defined in the text.

The coefficient β can be chosen in different ways.

Loop's original proposal was $\beta = \frac{1}{k}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{k}\right)^2\right)$. In particular, this is $\beta = 3/16$ for $k = 3$, and $\beta = 1/16$ for $k = 6$. Warren proposed later to keep $\beta = 3/16$ for $k = 3$, but to have simply $\beta = 3/8k$ for $k > 3$, which also amounts to $\beta = 1/16$ for $k = 6$ [87].

Edge rules are more complex. The rule for an edge point inserted on an edge *e* is chosen depending on the tag of the edge and the tags of adjacent vertices and sectors. There are several cases:

a) In the absence of tags, the standard edge rules are applied (Figure 7c).

b) If a crease edge *e* is split, and no endpoint is a dart vertex, the rule shown in Figure 7d is used.

c) If one of the endpoints is a dart vertex, the standard rule (Figure 7c) is applied.

d) If one endpoint *v* is an extraordinry vertex (valence not 4) and tagged as corner or smooth crease and the edge *e* is not a crease edge, we use a modified rule shown in Figure 7e (with weight $3/4 - \gamma$ associated with extraordinary vertex), with parameter γ chosen as $\gamma(\theta_k) = 1/2 - 1/4\cos\theta_k$. For a crease vertex, $\theta_k = \pi/k$, where *k* is the crease degree of the vertex in the sector where the edge *e* is.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

e) If both endpoints are extraordinary vertices (valence not 4) and tagged as corner or smooth crease and the edge e is not a crease edge, then the average of the masks given by the rules in 4 above is taken.

At a corner vertex *v*,ψ*e* is differentiated whether it is in a convex or concave sector. For a convex corner, $\theta_k = \alpha / k$, where α $\psi$is the angle between the two crease edges spanning the sector, for concave corners, $\theta_k = (2\pi - \alpha)/k$.

For concave corners, one may use $\theta_k > \pi / k$ and for convex corners $\theta_k < \pi / k$. The simplest choice of θ_k is $3\pi/(2k)$ for concave corners and $\pi/(2k)$ for convex corners which is the default; the best choice is to use α / k for convex corners (respectively $(2\pi - \alpha)/k$ for concave corners) where α is the angle between edges of the control mesh bounding the sector; θ_k for concave and convex corners can be also a user-defined parameter changing in the ranges specified above.

Application of the stencils is followed by application of flatness modification described in the next subclause. This modification is optional for all cases excluding concave corner; in which case it is necessary to ensure tangent plane continuity of the surface.

(i) Flatness modification

With *k* defined as in previous subclause, let the vector of control points be $p^m = \{p_c^m, p_0^m, \ldots, p_{k-1}^m\}$ (Figure 8). The superscript indicates the subdivision level the subscript indicates the point number in the sector. After subdivision step in a neighborhood of a corner vertex is modified as follows:

$$p^{new} = (1-s)p + s(\mathbf{a}_0 x^0 + \mathbf{a}_1 x^1 + \mathbf{a}_2 x^2)$$

where *s* is the flatness parameter. $\mathbf{a}_i = (l^i, p) \backsim l^0, l^1, l^2$ are limit position and tangent masks defined in the next subclause, and x^0, x^1, x^2 are the right eigenvectors also defined in the next subclause. The valid range of the flatness parameter is $0 \le s \le 1$ for all vertex types except concave corner, in which case it is $1 - \frac{1}{2\lambda} < s \le 1$, with lambda defined as $\lambda = 1/2 - 1/4(\cos\theta_k - \cos(\pi/k))$. Geometrically, the modified rule blends between control point positions before flatness modification and certain points in the tangent plane, which are typically close to the projection of the original control point. The limit position $\mathbf{a}_0$ of the center vertex remains unchanged.

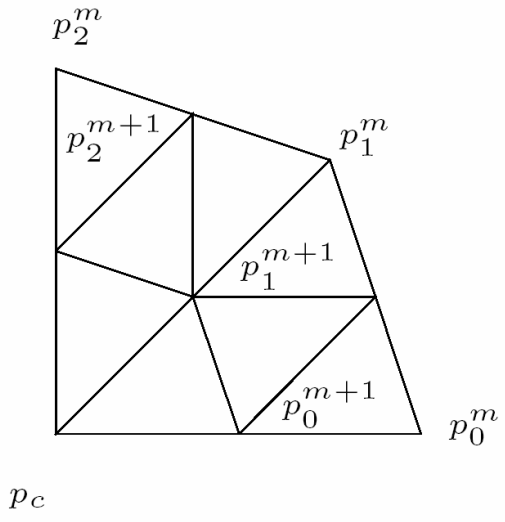


Figure 8 — Neighborhoods of a vertex on different subdivision levels. The subdivision matrix relates the vector of control points p^m to the control points on the next level p^{m+1}. For a neighborhood of *k* triangles $p^m = \{p_c^m, p_0^m, \ldots, p_{k-1}^m\}$.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The flatness modification is always applied at concave corner vertices; the default value for the flatness parameter is 1. Better visual results are obtained for $s = 1 - 1/(4\lambda)$. In other cases, s can be taken to be 0 by default and the modification step need not be applied.

Optionally, a user-specified flatness value can be associated with any vertex and used to control how fast the surface approaches the tangent plane.

A compliant implementation is required to support *s =0* and *s = 1*, but an implementation may support arbitrary *s* values.

(ii) Limit position and tangent masks

For each type of vertex (interior, smooth crease, corner, dart) there are masks for computing the limit position of this vertex and two tangent vectors at this vertex. The limit position masks are denoted l^0 and limit tangent position masks are denoted l^1 and l^2 respectively. A subscript *c*ψdenotes the coefficient corresponding to the center vertex. Once two tangent vectors are obtained by applying the masks, the normal can be computed as the cross-product of the two. In addition we define right eigenvectors in each case which are necessary for flatness modification. Recall that dominant right eigenvector x^0 is the vector consisting of ones

Untagged interior or dart vertex of degree *k*. In all cases *i* is in the range $0 \ldots k-1$, and $\theta_k = 2\pi / k$.

$$l_c^0 = \frac{1}{1+(8k/3)\beta}, \quad l_i^0 = \frac{(8/3)\beta}{1+(8k/3)\beta}$$

$$x_c^1 = x_c^2 = l_c^1 = l_c^2 = 0$$

$$x_i^1 = \sin i\theta_k, \quad x_i^2 = \cos i\theta_k,$$

$$l_i^1 = \frac{2}{k}\sin i\theta_k, \quad l_i^2 = \frac{2}{k}\cos i\theta_k,$$

Smooth crease vertex of crease degree *k*. Let $\theta_k = \pi / k$

$$l_c^0 = 2/3, \quad l_1^0 = l_k^0 = 1/6, \quad l_i^0 = 0, \quad i = 1 \ldots k-1$$

For *k=1*,

$$x_c^1 = -1/3, x_1^1 = x_2^1 = 2/3, x_c^2 = 0, x_1^2 = 1, x_2^2 = -1$$

$$l_c^1 = -1, l_1^1 = l_2^1 = 1/2, l_c^2 = 0, l_1^2 = 1/2, l_2^2 = -1$$

Otherwise $x_c^1 = x_c^2 = l_c^1 = 0$,

$$x_i^1 = \sin i\theta_k, x_i^2 = \cos i\theta_k, i = 0 \ldots k$$

$$l_0^1 = 1/2, l_k^1 = -1/2, l_i^1 = 0, i = 1 \ldots k$$

$$l_c^2 = -\frac{2}{k}\left(\left(\frac{2}{3} - a\right)\sigma_1 - b\sigma_3\right)$$

$$l_0^2 = l_k^2 = -\frac{2}{k}\left(\left(\frac{a}{2} + \frac{1}{6}\right)\sigma_1 + \frac{1}{2}b\sigma_3\right)$$

$$l_i^2 = \frac{2}{k}\sin i\theta_k, \quad i = 1 \ldots k-1$$

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

where

$$a = \frac{\frac{1}{4}(1+\cos\theta_k)}{3(\frac{1}{2}-\frac{1}{4}\cos\theta_k)} \qquad b = \frac{\frac{2}{3}-a}{\cos\frac{k\zeta}{2}}$$

$$\sigma_1 = \frac{\sin\theta_k}{1-\cos\theta_k} \qquad \sigma_3 = \frac{\cos\frac{k\zeta}{2}\sin\theta_k}{\cos\zeta-\cos\theta_k}$$

$$\zeta = \arccos(\cos\theta_k - 1)$$

Convex/concave corner vertex of crease degree *k* with parameter θ_k . Let $\theta = k\theta_k$.

$$l_c^0 = 1, l_i^0 = 0, i = 0 \ldots k$$

$$x_c^1 = x_c^2 = 0, x_i^1 = \frac{\sin i\theta_k}{\sin\theta}, x_i^2 = \frac{\sin(k-i)\theta_k}{\sin\theta}, i = 0 \ldots k$$

$$l_c^1 = -1, l_k^1 = 1, l_i^1 = 0, i = 0 \ldots k-1$$

$$l_c^2 = -1, l_0^2 = 1, l_i^2 = 0, i = 1 \ldots k$$

4.3.3.2.3.2 Extended Loop subdivision

Extended Loop is an enhancement of the well-known triangle-based Loop subdivision algorithm that allows the designer to work with quadrilateral-based models which are better suited to capturing the symmetries of natural and man-made objects, while ensuring that the visualisation process can be optimised for a triangle-only rendering pipeline. Furthermore, unlike quadrilateral-based subdivision schemes such as Catmull-Clark, extended Loop offers the model designer exact control over the final model triangulation, which is essential for efficient rendering of low polygon models.

(i) Stencils

Extending Loop to handle arbitrary meshes requires the addition of edge visibility tags for edges that are introduced during triangulation and are not part of the original control hull. The introduction of edge visibility information will clearly affect the vertex and edge subdivision rules.

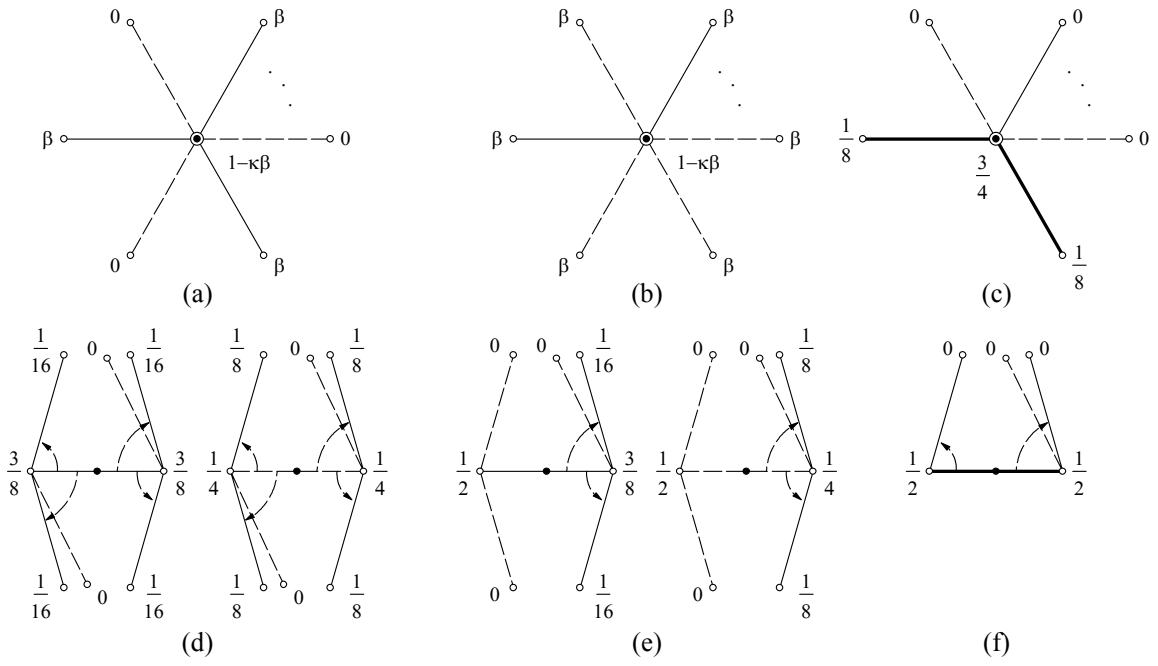


Figure 9 — Stencils of extended Loop subdivision for repositioning interior(smooth and dart) (a), (b) and boundary/crease old vertices and splitting interior (d), (e) and boundary/crease edges.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Figure 9 shows the extended Loop subdivision stencils used for repositioning old vertices (as with standard Loop, corner vertices are never moved) and splitting edges both on the surface interior and boundaries/creases. Solid lines indicate original control hull (visible) edges while dashed lines indicate invisible edges. It is clear from the stencils (a), (b) and (c) that repositioning old vertices with extended Loop is identical to standard Loop if invisible edges are ignored, with one exception (b). If a vertex has two or less visible edges in its neighborhood, then the invisible edges are included in the calculation. If all edges are visible then the stencils are identical to standard Loop subdivision.

As with standard Loop subdivision, $\beta = \frac{1}{k}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{k}\right)^2\right)$, where k is the number of visible edges (Figure 9a) or the valence (visible + invisible edges) if the number of visible edges is ≤ 2 (Figure 9b).

The stencil for splitting boundary/crease edges (Figure 9f) is also identical to standard Loop, while splitting interior edges depends on whether the edge in question is visible (Figure 9d – left) or not (Figure 9d – right). As with the old vertex repositioning stencils, the edge split stencils try to employ vertices at the "other end" of visible edges only, ignoring invisible edges completely. Identifying visible edges at each side of the edge endpoint involves a clockwise and/or counter-clockwise search around the endpoint neighborhood and may result in the same vertex for both sides or no visible edges in which case the endpoint is assigned the full weight (Figure 9e). Again if all edges are visible, the stencils are identical to standard Loop subdivision.

(ii) Visibility propagation

The visibility of the new edges needs to be determined, since it will be required on the next round of subdivision. New edges radiate from a new point (see Figure 10), and their visibility is determined according to the neighborhood of the point.

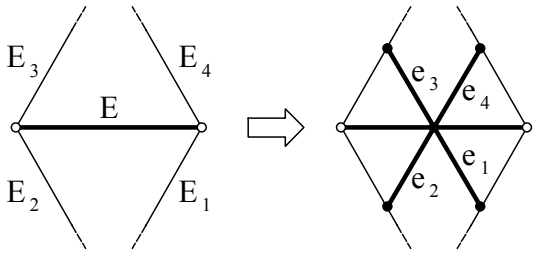


Figure 10 — New edges created when splitting an edge in two.

Two edges will be the two halves of the original edge E - these simply keep the visibility of the original edge. Of the other four new edges, two edges will be to the right hand side of the original edge, and two more to its left. If we label these edges clockwise as e_1, e_2, e_3 and e_4, these connect to other new points created by dividing the original edges labeled E_1, E_2, E_3 and E_4. The visibility of the new edges is initialized to invisible and is changed to visible if the lookup of the combination of E, E_1, E_2, E_3 and E_4 in Table 1 indicates that it should be. Clearly, the visibility of each of e_1, e_2, e_3 and e_4 is not uniquely determined by splitting E alone; e_1, for example, also depends on the visibility lookup for splitting E_1.

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1 — New edge visibility lookup table.

Original edges visible					New edges visible			
E	E_1	E_2	E_3	E_4	e_1	e_2	e_3	e_4
				✓	✓		✓	
			✓			✓		✓
			✓	✓	✓	✓		
		✓			✓		✓	
		✓		✓	✓		✓	
		✓	✓		✓			✓
		✓	✓	✓	✓			
	✓					✓		✓
	✓			✓		✓	✓	
	✓		✓			✓		✓
	✓		✓	✓		✓		
	✓	✓					✓	✓
	✓	✓		✓			✓	
	✓	✓	✓					✓
	✓	✓	✓	✓	✓	✓	✓	✓

Original edges visible					New edges visible			
E	E_1	E_2	E_3	E_4	e_1	e_2	e_3	e_4
✓								
✓				✓				
✓			✓					
✓			✓	✓			✓	✓
✓		✓						
✓		✓		✓				
✓		✓	✓					
✓		✓	✓	✓			✓	✓
✓	✓							
✓	✓			✓				
✓	✓		✓					
✓	✓		✓	✓			✓	✓
✓	✓	✓			✓	✓		
✓	✓	✓		✓	✓	✓		
✓	✓	✓	✓		✓	✓		
✓	✓	✓	✓	✓	✓	✓	✓	✓

Note that Table 1 is unchanged if we swap left and right, so it is unimportant which side of the original edge we call “right” to start with.

(iii) Vertex normal calculation

In order to perform operations such as lighting on a surface, it is necessary to determine the normal to that surface at each vertex. This is the direction at right angles to the limit surface at that point. In order to calculate this normal, it is necessary first to obtain two tangent vectors, t_1 and t_2, which are directions that lie along the surface rather than at right angles to it. Given these, a vector cross product will result in the desired normal:

$$n = t_1 \times t_2 .$$

© ISO/IEC 2006 – All rights reserved

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

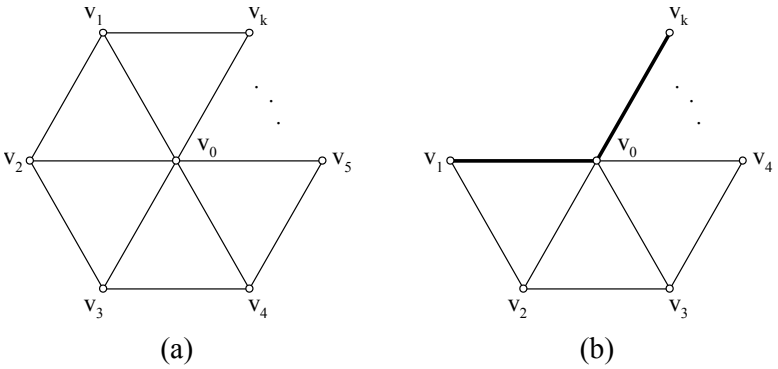


Figure 11 — (a) interior, and (b) boundary/crease vertex neighborhood used for normal calculation.

For interior smooth and dart vertices of valence *k* (Figure 11a), the tangents are defined as the weighted averages of the positions of the neighborhood vertices (v_i where $i \in [1, k]$):

$$\begin{aligned} t_1 &= c_1 v_1 + c_2 v_2 + c_3 v_3 + \ldots + c_k v_k \\ t_2 &= c_2 v_1 + c_3 v_2 + c_4 v_3 + \ldots + c_k v_{k-1} + c_1 v_k \end{aligned},$$

where:

$$c_i = \cos\frac{2\pi i}{k}.$$

For boundary, crease and corner vertices (Figure 11b), each sector between pairs of boundary/crease edges will have a different pair of tangent vectors that are defined as:

$$t_1 = v_1 - v_k$$

$$t_2 = \begin{cases} -2v_0 + v_1 + v_2, & k = 2 \\ v_2 - v_0, & k = 3 \\ -2v_0 - v_1 + 2v_2 + 2v_3 - v_4, & k = 4 \\ w_1 v_1 + w_2 v_2 + w_3 v_3 + \ldots + w_k v_k, & k > 4 \end{cases},$$

where, for $\theta = \frac{\pi}{k-1}$, w_i is defined as:

$$w_i = \begin{cases} \sin\theta, & i = 1, i = k \\ (2\cos\theta - 2)\sin((i-1)\theta), & 2 \le i \le k-1 \end{cases}.$$

4.3.3.2.3.3 Catmull-Clark subdivision

The Catmull-Clark scheme was described in [22]. This scheme can be applied to general polygonal meshes, but after the first subdivision step all polygons in a mesh become quads. The masks for a mesh consisting only of quads are shown in Figure 12 and Figure 13. The scheme produces surfaces that are C^2 everywhere except at extraordinary vertices, where they are C^1.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

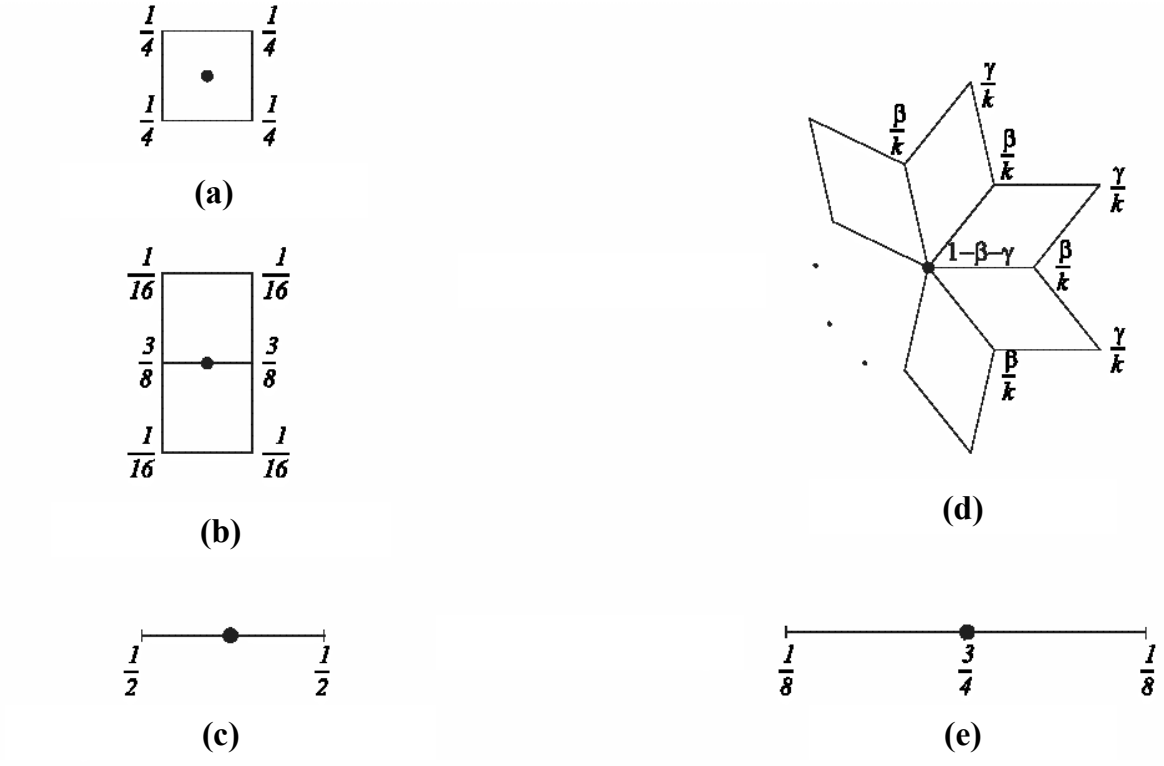


Figure 12 — Catmull-Clark rules with *k* defined as in subclause 4.3.3.1.2 (a) face rule; (b) regular edge rule (c) boundary/crease edge rule (d) interior vertex rule $\beta = \frac{3}{2k}$ and $\delta = \frac{1}{4k}$ (e) smooth boundary/crease vertex rule

There are three types of rules: vertex rules used to update old vertex positions, edge rules used to insert new vertices for each edge, and face rules used to insert new vertices for each face.

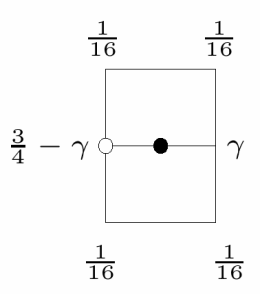


Figure 13 — Modified rule for odd vertices adjacent to a boundary extraordinary vertex (Catmull-Clark scheme), the coefficient gamma is defined in the text.

The rules used only on the first step are:

— a face control point for an *n*-gon is computed as the average of the corners of the polygon;

— an edge control point for an untagged edge as the average of the endpoints of the edge and newly computed face control points of adjacent faces; for a crease/boundary edge it is the average of the endpoints;

© ISO/IEC 2006 – All rights reserved

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

— The vertex rule for untagged interior and dart vertices is:

$$p_c^{j+1} = \frac{k-2}{k} p_c^j + \frac{1}{k^2}\left(\sum_{i=0}^{k-1} p_i^j + q_i^{j+1}\right)$$

where p_i^j are *k* control points at level *j* sharing edges with p_c^j, and q_c^{j+1} are face control points for faces incident at the vertex. Note that face control points on level $j-1$ are used.

— For boundary/crease vertices, it is the spline rule identical to the one used for the Loop scheme. The corner vertex positions are never changed.

After the first subdivision step the mesh consists only of quadrilaterals and the subdivision rules are described in (Figure 12 and Figure 13). The vertex and face rules are essentially the same; the masks are shown in Figure 12. As in the case of the Loop scheme edge rules are more complex:

— In the absence of tags, the standard edge rules are applied (Figure 14b).

— If a crease edge *e* is split and no endpoint is tagged as a dart vertex , rules of Figure 14c are used.

— If one of the endpoints is a dart vertex, the standard edge rule is used (Figure 14b).

— If one endpoint *v* is tagged as a corner or smooth crease and the edge *e* is not a crease, a modified rule shown in Figure 6 is used, with parameter γ chosen as $\gamma(\theta_k) = 3/8 - 1/4\cos\theta_k$. Exactly like for Loop scheme, for a smooth crease vertex, $\theta_k = \pi/k$, where *k* is the crease degree of *v* in the sector where *e* is. For a corner use $\theta_k = \alpha/k$, where $\alpha > \pi$ for concave corners and $\alpha < \pi$ for convex corners.

— If both endpoints are tagged as corner or smooth crease, then the average of the masks given by the rules above is taken.

Application of the stencils is followed by application of flatness modification described in the next subclause. This modification is optional for all cases excluding concave corner; in this case it is necessary to ensure tangent plane continuity of the surface.

(i) Flatness modification

The flatness modification is similar to the one used for Loop scheme, but the vector of control points has to include one more group of points $[q_0^m, \ldots, q_{k-1}^m]$ as shown in Figure 14. The superscript indicates the subdivision level the subscript indicates the point number in the sector.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

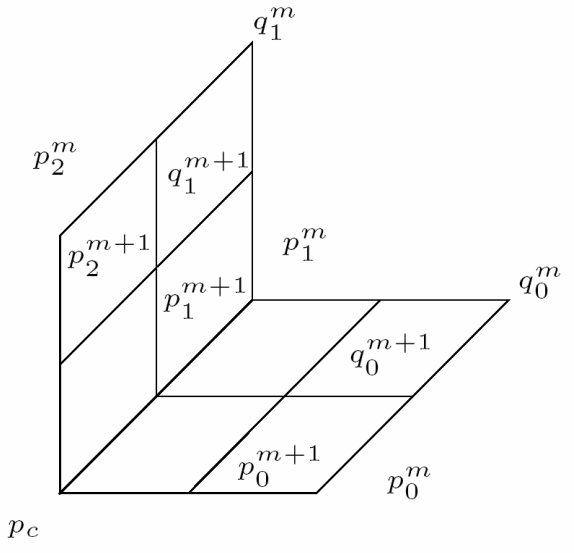


Figure 14 — Neighborhoods of a vertex on different subdivision levels. The subdivision matrix relates the vector of control points p^m to the control points on the next level p^{m+1}. For a neighborhood of *k* quadrilaterals $p^m = \{p_c^m, p_0^m, \ldots, p_{k-1}^m, q_0^m, \ldots, q_{k-1}^m\}$.

The rule itself coincides with the rule for the Loop scheme:

$$p^{new} = (1-s)p + s(\mathbf{a}_0 x^0 + \mathbf{a}_1 x^1 + \mathbf{a}_2 x^2)$$

where *s* is the flatness parameter. $\mathbf{a}_i = (l^i, p) \sim l^0, l^1, l^2$ are limit position and tangent masks defined in the next subclause, and x^0, x^1, x^2 are the right eigenvectors also defined in the next subclause.

The valid range of the flatness parameter is $0 \le s \le 1$ for all vertex types except concave corner, in which case it is $1 - \frac{1}{2\lambda} < s \le 1$, with lambda defined as

$$\lambda = \frac{1}{2}\gamma + \frac{1}{8}c^2 + \frac{1}{16}\sqrt{64\gamma^2 + 32c^2\gamma - 48\gamma + 4c^4 + 4c^2 + 9} \text{ and } c = \cos\frac{\pi}{2k}.$$

The default flatness parameter can be set to zero for all cases except concave corners, in which case it is set to 1. Better visual results are obtained for the values close to $s = 1 - 1/(4\lambda)$.

Optionally, a user-specified flatness value can be associated with any vertex and used to control how fast the surface approaches the tangent plane.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(ii) Limit position and tangent masks

Interior or dart vertex of degree *k*. In all cases *i* is in the range $0 \ldots k-1$, and $\theta_k = 2\pi / k$.

$$l_c^0 = \frac{k}{k+5}, \quad l_{pi}^0 = \frac{4}{k(k+5)}, \quad l_{qi}^0 = \frac{1}{k(k+5)}$$

$$x_c^1 = x_c^2 = l_c^1 = l_c^2 = 0$$

$$x_{pi}^1 = \frac{1}{\sigma}\sin i\theta_k \quad x_{pi}^2 = \frac{1}{\sigma}\cos i\theta_k$$

$$x_{qi}^1 = \frac{\sin i\theta_k + \sin(i+1)\theta_k}{\sigma(4\lambda-1)}, \quad x_{qi}^2 = \frac{\cos i\theta_k + \cos(i+1)\theta_k}{\sigma(4\lambda-1)},$$

$$l_{pi}^1 = 4\sin i\theta_k, \quad l_{pi}^2 = 4\cos i\theta_k,$$

$$l_{qi}^1 = \frac{\sin i\theta_k + \sin(i+1)\theta_k}{4\lambda-1}, \quad l_{qi}^2 = \frac{\cos i\theta_k + \cos(i+1)\theta_k}{4\lambda-1}$$

where $\lambda = 5/16 + 1/16(\cos\theta_k + \cos\theta_k / 2\sqrt{9+\cos 2\theta_k})$ and $\sigma = k(2 + \frac{1+\cos\theta_k}{(4\lambda-1)^2})$

Smooth crease vertex of crease degree *k*. Let $\theta_k = \pi / k$

$$l_c^0 = 2/3, \quad l_{p1}^0 = l_{pk}^0 = 1/6,$$

$$l_{q0}^0 = l_{qi}^0 = l_{pi}^0 = 0, \quad i = 1 \ldots k-1$$

For $k = 1$,

$x_c^1 = 1/18$	$x_{p0}^1 = -2/18$	$x_{p1}^1 = -2/18$	$x_{q0}^1 = -5/18$
$x_c^2 = 0$	$x_{p0}^2 = -1/2$	$x_{p1}^2 = 1/2$	$x_{q0}^2 = 0$
$l_c^1 = 6$	$l_{p0}^1 = -3$	$l_{p1}^1 = -3$	$l_{q0}^1 = 0$
$l_c^2 = 0$	$l_{p0}^2 = -1$	$l_{p1}^2 = 1$	$l_{q0}^2 = 0$

Otherwise $l_c^2 = x_c^1 = x_c^2 = 0$,

$$x_{pi}^1 = \sin i\theta_k, \; x_{pi}^2 = \cos i\theta_k, \; i = 0 \ldots k$$

$$x_{qi}^1 = \sin i\theta_k + \sin(i+1)\theta_k, \; i = 0 \ldots k-1$$

$$x_{qi}^2 = \cos i\theta_k + \cos(i+1)\theta_k, \; i = 0 \ldots k-1$$

$$l_{p0}^2 = 1/2, l_{pk}^2 = -1/2, l_{q0}^2 = l_{qi}^2 = l_{pi}^2 = 0, i = 1 \ldots k-1$$

$$l_c^1 = 4R(\cos\theta_k - 1), l_{p0}^1 = l_{pk}^1 = -R(1 + 2\cos\theta_k)$$

$$l_{pi}^1 = \frac{4\sin i\theta_k}{(3+\cos\theta_k)k}, i = 1 \ldots k-1$$

$$l_{qi}^1 = \frac{4(\sin i\theta_k + \sin(i+1)\theta_k)}{(3+\cos\theta_k)k}, i = 0 \ldots k-1$$

where $R = \dfrac{\cos\theta_k + 1}{k\sin\theta_k(3+\cos\theta_k)}$

Convex/concave corner vertex of crease degree *k* with parameter θ_k. Let $\theta = k\theta_k$.

Left eigenvectors are the same as for Loop with zeroes everywhere except l_c, l_{p0} and l_{pk}.

$$x_{pi}^1 = \frac{\sin i\theta_k}{\sin\theta}, \; x_{pi}^2 = \frac{\sin(k-i)\theta_k}{\sin\theta}, \; i = 0 \ldots k$$

$$x_{qi}^1 = \frac{\sin i\theta_k + \sin(i+1)\theta_k}{\sin\theta}, \; i = 0 \ldots k-1$$

$$x_{qi}^2 = \frac{\sin(k-i)\theta_k + \sin(k-i-1)\theta_k}{\sin\theta}, \; i = 0 \ldots k-1$$

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.3.3.2.3.4 Normal control

In addition to the flatness modification which insures that the surface is tangent plane continuous at concave corner vertices and provides additional shape control at other vertex types, a similar modification mechanism can be used to control the tangent plane of the surface as shown in Figure 15.

Specifically, if the user prescribes tangent vectors or the normal for a sector at a corner or smooth boundary vertex or a normal for an interior vertex then the following modification can be applied after each subdivision step for both Loop and Catmull-Clark surfaces,

$$p^{new} = p + t((\mathbf{a}'_1 - \mathbf{a}_1)x^1 + (\mathbf{a}'_2 - \mathbf{a}_2)x^2)$$

where the notation used is the same as in the subclauses describing flatness modification above, $\mathbf{a}'_1$ and $\mathbf{a}'_2$ denote the prescribed tangents and t is a user-specified parameter $0 \le t \le 1$, by default set to 1. If only the normal is prescribed, the tangents are computed as $\mathbf{a}'_i = \mathbf{a}_i - (\mathbf{a}_i, n)n \quad i = 1,2$ -

The parameter t determines how fast the surface approaches the prescribed tangent plane.

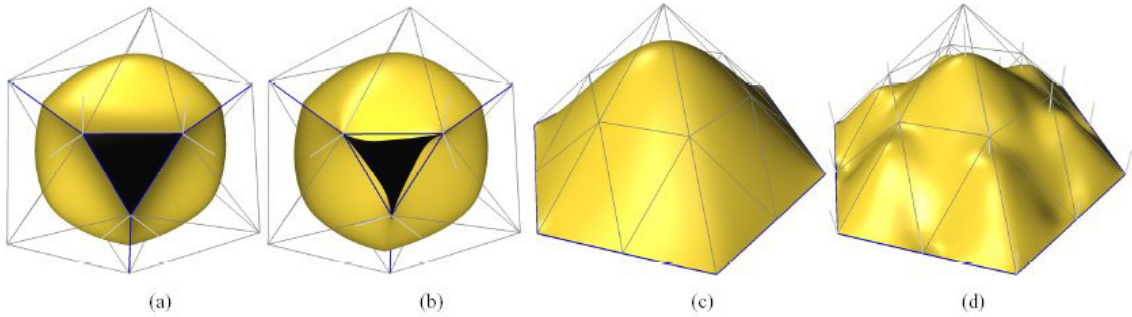


Figure 15 — Normal interpolation. (a) Surface with convex corners. (b) Prescribed directions: at each corner we tilt the normal for one surface sector slightly inwards. (c) Smooth surface. (d) Same control mesh but all normals vertical.

4.3.3.3 SubdivSurfaceSector

4.3.3.3.1 Node interface

```
SubdivSurfaceSector { #%NDT=SFSubdivSurfaceSectorNode
    field           SFInt32     faceIndex       0   # face index from coordIndex
    field           SFInt32     vertexIndex     0   # vertex index from coord
    exposedField    SFInt32     tag             0
    exposedField    SFFloat     flatness        0
    exposedField    SFFloat     theta           0
    exposedField    SFVec3f     normal          0 0 0
    exposedField    SFFloat     normalTension   0
}
```

4.3.3.3.2 Functionality and semantics

The **faceIndex** field contains the index of the face and the **vertexIndex** field contains the vertex index the sector is attached to. The indices refer to point index and face index from **coord** and **coordIndex** fields respectively in the parent **SubdivisionSurface** node.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **tag** field indicates the type of sector (Figure 16):

Tag value	Semantic
0	No corner sector
1	Convex sector
2	Concave sector

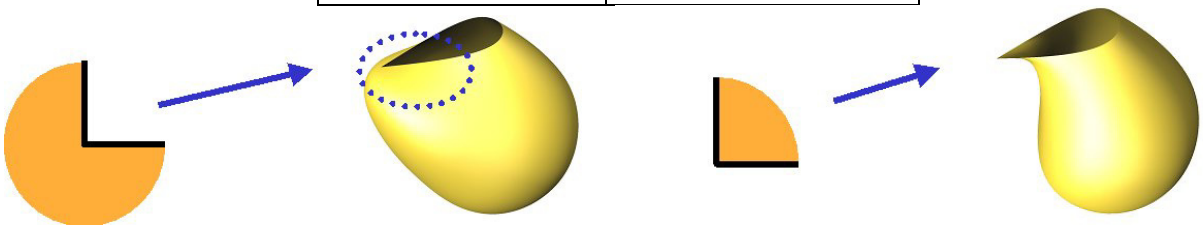


Figure 16 — Concave corners (left) and convex corners (right).

The **flatness** field contains a value between 0 and 1: 0 indicating no modified flatness, and 1 indicating maximal flatness. This parameter is denoted *s* in Loop and Catmull-Clark rule definitions above (Figure 17).

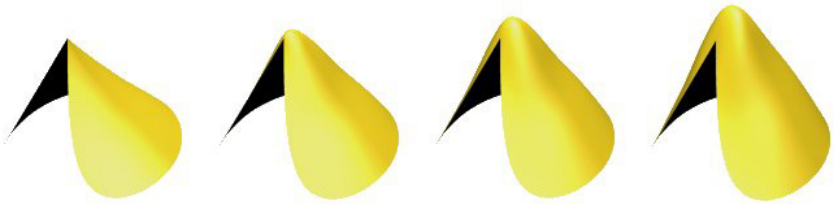


Figure 17 — Influence of the flatness parameter.

The **theta** field is an angle used to control the shape of the corner; this angle is denoted θ_k in the mask definitions above (Figure 18).

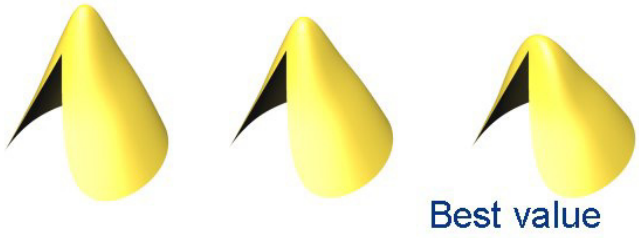


Figure 18 — Influence of theta.

The **normal** field specifies the desired normal at the vertex (Figure 19).

The **normalTension** field is the influence of the normal between 0 and 1, 0 meaning no influence; denoted *t* in the formulas for normal modification (Figure 19).

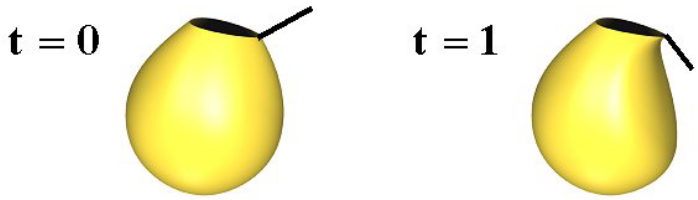


Figure 19 — Prescribed normals and tension parameter.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2006 – All rights reserved

4.3.3.4 WaveletSubdivisionSurface

4.3.3.4.1 Node interface

```
WaveletSubdivisionSurface { #NDT=%SF3DNode
   exposedField   SFGeometryNode     baseMesh      NULL
   exposedField   SFFloat            frequency     1.0
   exposedField   SFFloat            fieldOfView   0.785398    # pi/4
   exposedField   SFInt32            quality       1
}
```

4.3.3.4.2 Functionality and semantics

Wavelet representation of surfaces relies on an extension of classical multiresolution analysis theory in which operators similar to those of subdivision surfaces are used in place of traditional interval subdivision and low-pass filtering (scaling spaces).

In this representation, a 3D mesh is received as a coarse mesh and independent sets of zero-tree encoded wavelet coefficients, allowing adaptive and progressive reconstruction at the client side.

The **WaveletSubdivisionSurface** node is used to represent adaptively multi-resolution polygonal models. It allows representing at any moment a 3D-model at a resolution that depends on the distance of the object with respect to the viewpoint. Transmission of such a model consists of two parts: a base mesh, which describes the model at the coarsest level of detail, and series of refinements (wavelet coefficients) that can be carried on one or more elementary streams (see the bitstream definition in subclause 5.3.1).

The **baseMesh** field shall specify either a **SubdivisionSurface** node or an **IndexedFaceSet**. If **baseMesh** is a **SubdivisionSurface**, the stream containing the mesh shall consist of a single access unit, and it will be explicitly instantiated as a **SubdivisionSurface**. The structure of this **SubdivisionSurface** will be preserved, namely deletion and insertion of new faces and/or vertices is explicitly forbidden. However, replacing the values of the coordinates is allowed. If **baseMesh** is an **IndexedFaceSet**, it is required to be manifold.

NOTE — Instantiating explicitly **baseMesh** as a **SubdivisionSurface** or **IndexedFaceSet** allows animating the node.

The terminal is responsible for maintaining an internal representation of the mesh to be rendered consisting of the **baseMesh** to which the **downstream** data must be applied.

The **frequency** field gives a preferred frequency at which the terminal shall give its state by the means of the back channel stream if it exists.

The **quality** field allows parameterizing the quality of the rendered object. The range of values for this field is [0…2] where 0 stands for low quality, 1 for normal quality, 2 for best quality. Those values don't have a more explicit semantic, that is, it is a hint for the terminal.

The player sees the **WaveletSubdivisionSurface** node as a **SubdivisionSurface** or **IndexedFaceSet** node container. That is, the player has access to the **baseMesh** field.

4.3.4 MeshGrid representation

4.3.4.1 Introduction

The peculiarity of the MeshGrid representation [71] is that it defines the object's wireframe (see e.g. Figure 20(a)) by (1) describing the connectivity between the vertices located on the surface of the object by a so-

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

© ISO/IEC 2006 – All rights reserved

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

called connectivity-wireframe (see Figure 20(b)), and (2) positioning these vertices in relation to a regular 3-D grid of points, i.e. the reference-grid (see Figure 20(c)).

Both the connectivity-wireframe and the reference-grid can be represented hierarchically, as shown in Figure 20(d) and Figure 20(e) respectively for 3 levels in the hierarchy.

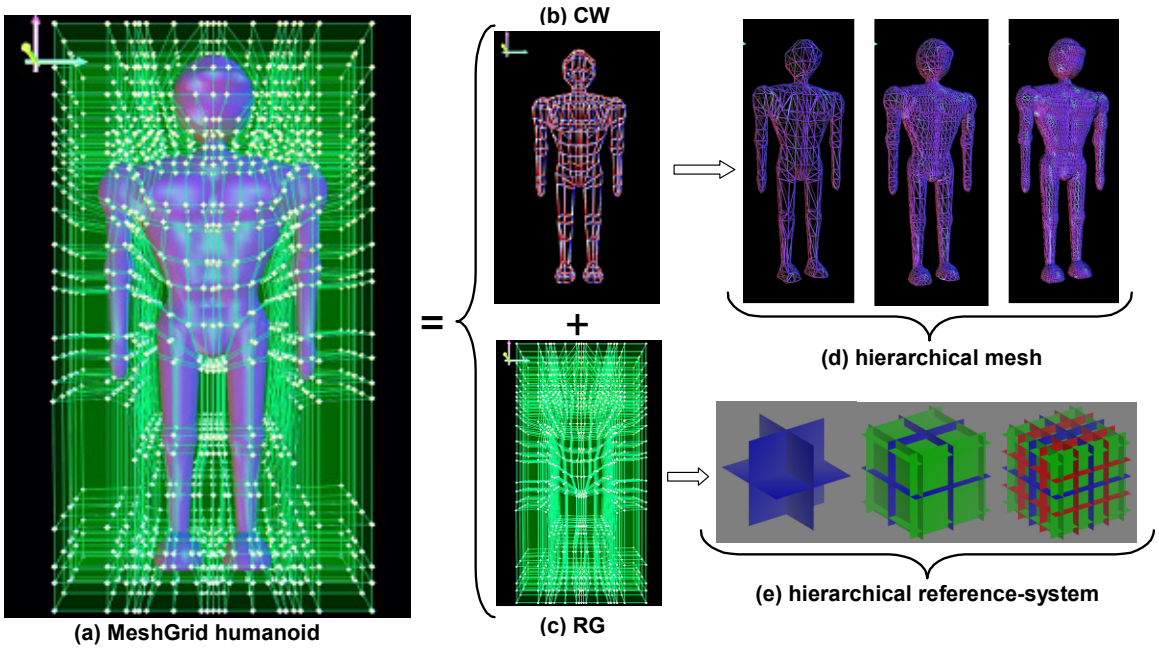


Figure 20 — A MeshGrid humanoid model designed by means of implicit surfaces.

4.3.4.1.1 The reference-grid (RG) – the reference-system of the MeshGrid model.

The reference-grid (see Figure 20(c)) is the reference system upon which the MeshGrid model is built, and is defined by the intersection points between three sets of reference-surfaces S_U, S_V, S_W as given by the formula (1)

$$RG = \bigcap \left\{ \sum_U S_U, \sum_V S_V, \sum_W S_W \right\}. \qquad (1)$$

The discrete position (u,v,w) of a reference-grid point represents the indices of the reference-surfaces $\{S_U, S_V, S_W\}$ intersecting in that point, while the coordinate (x,y,z) of a reference-grid point is equal to the coordinate of the computed intersection point.

A multi-resolution model is designed by choosing a hierarchical reference-system, the reference-surfaces belonging to the same resolution level being displayed in the same color in the example from Figure 20(e).

The reference system should be chosen according to the topology of the object, i.e. it has higher density in (1) areas where the curvature is high, or (2) places that are deformed during the animation. For the humanoid model of Figure 20, the reference system is adjusted for traversing the anatomical articulations (joints) of the body, in order to be able to virtually split the resulting seamless connectivity-wireframe into meaningful anatomical parts (such as the shoulder, elbow, wrist).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.3.4.1.2 The connectivity-wireframe (CW) – the surface mesh.

The connectivity-wireframe (see Figure 20(b)) is generated by merging the contouring of the original 3D model in each of the reference-surfaces S_U, S_V, S_W as given by the formula (2)

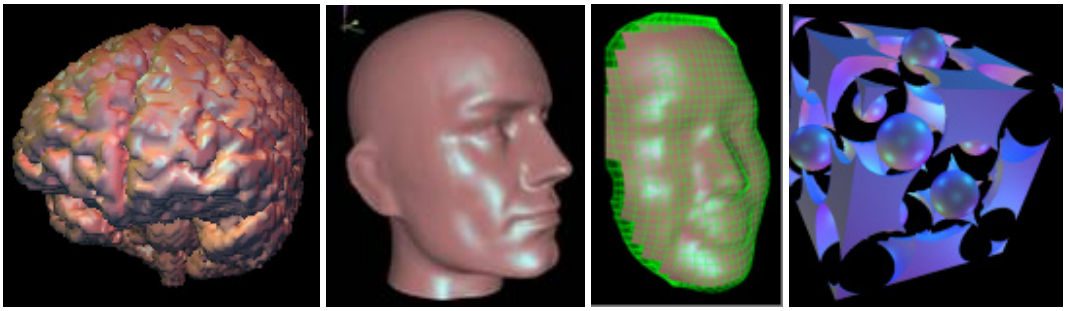
$$CW = \bigcup \left\{ \sum_U C(S_U), \sum_V C(S_V), \sum_W C(S_W) \right\}, \qquad (2)$$

in which $C(S_x)$ represents the contour obtained by the intersection with surface S_x. Each vertex is located at the intersection point between two contours.

Subdividing the reference surfaces hierarchically, as explained in subclause 4.3.4.1.1, results in a multi-resolution connectivity-wireframe, in which each resolution level consists of a single seamless mesh, as shown in Figure 20(d).

4.3.4.1.3 The features

This type of representation is specific for describing objects defined as series of contours or slices, such as: discrete 3D data sets (e.g. 3D medical images, processed range scanner data), cylindrical and spherical projected scanner meshes based on quadrilaterals, or generic models (see Figure 21(a)-(d)).



(a) (b) (c) (d)

Figure 21 — The rendered surface obtained from (a) a 3D voxel model of the brain, (b) a structured light scanner quadrilateral mesh, respectively with its deformed reference-grid in overlay in (c), and (d) a generic model.

A MeshGrid mesh allows view-dependent streaming (see subclause 5.2) and particular animation possibilities, such as: hierarchical grid animation, and offset based animation (see subclause 4.3.4.3).

4.3.4.2 MeshGrid

4.3.4.2.1 Node specification

MeshGrid {

eventIn	MFInt32	**set_colorIndex**	
eventIn	MFInt32	**set_coordIndex**	
eventIn	MFInt32	**set_normalIndex**	
eventIn	MFInt32	**set_texCoordIndex**	
exposedField	SFColorNode	**color**	NULL
exposedField	SFCoordinateNode	**coord**	NULL
exposedField	SFInt32	**displayLevel**	0
exposedField	SFInt32	**filterType**	0
exposedField	SFCoordinateNode	**gridCoord**	NULL
exposedField	SFInt32	**hierarchicalLevel**	0
exposedField	MFInt32	**nLevels**	[]
exposedField	SFNormalNode	**normal**	NULL

© ISO/IEC 2006 – All rights reserved

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

exposedField	MFInt32	**nSlices**	[]
exposedField	SFTextureCoordinateNode	**texCoord**	NULL
exposedField	MFInt32	**vertexLink**	[]
exposedField	MFFloat	**vertexOffset**	[]
field	MFInt32	**colorIndex**	[]
field	MFInt32	**coordIndex**	[]
field	MFInt32	**normalIndex**	[]
field	SFBool	**solid**	TRUE
field	MFInt32	**texCoordIndex**	[]
eventOut	SFBool	**isLoading**	
eventOut	MFInt32	**nVertices**	

}

4.3.4.2.2 Node semantics

The **MeshGrid** node is derived from the existing **IndexedFaceSet** node. The following fields have been removed, since the triangulation of the surface is pre-defined in the decoder: *ccw*, *convex*, *creaseAngle*. Some new fields are introduced, specific to the MeshGrid representation. The mesh is manifold, but can be open or closed.

The fields of the MeshGrid node can be encoded either (1) by BIFS, or (2) as a binary stream. When encoded as a binary stream, the following fields of the MeshGrid node will be initialized when decoding the binary stream (see subclause 5.2.2.36): **nSlices**, **nLevels**, **gridCoord**, **coord**, **coordIndex**, **vertexOffset**, **vertexLink**, **nVertices.** The **isLoading** flag is set to true while decoding the binary stream, respectively becomes false when the decoding finishes. The value of **isLoading** flag has to be checked before updating the fields in order to make sure that the decoder has finished modifying them. If encoded by BIFS only the **vertexLink** (the description of the connectivity-wireframe) and **vertexOffset** (the vertices' offsets relative to the reference-grid) fields need to be decoded, as explained in subclauses 5.2.3.1 and 5.2.3.3.

The **nSlices** field specifies the number of slices of the three sets $\{S_U,S_V,S_W\}$ of reference-surfaces defining the reference-grid at the last resolution level. The number of slices in the *U* direction equals to **nSlices[0],** respectively the number of slices in the *V* direction equals to **nSlices[1]**, and the number of slices in the *W* direction equals to **nSlices[2]**. The minimum number of slices in the *U* and *V* directions is "2" while in the *W* direction the minimum number of slices is "1", in which case the reference-grid is single layer. More detail can be found in subclauses 5.2.3.4 and B.1.1.

The **nLevels** field defines the number of resolution levels of the MESHGRID mesh. The number of resolution levels in the *U* direction equals to **nLevels[0]**, respectively the number of resolution levels in the *V* direction equals to **nLevels[1]**, and the number of resolution levels in the *W* direction equals to **nLevels[2].** More detail can be found in B.1.1.

The **gridCoord** field defines the reference-grid points. The number of reference-grid points can be either equal (1) to the number of reference-grid corners or (2) to the following value: $nSlices[0] \times nSlices[1] \times nSlices[2]$. In case (1), the values represent the coordinates of the reference-grid corners, and the decoder will generate a regular distributed reference-grid based on the coordinates of these corners. The number of corners can either be 1, 2, 4 or 8 depending on the type of model, i.e. single layer and cyclic, only single layer, or generic. In this case the **gridCoord** buffer is resized and its initial values are replaced by the computed coordinates of the grid points. In case (2) the values represent the coordinates of the reference-grid points. Any point from the reference-grid can be addressed by a distinct (*u*,*v*,*w*) discrete position, where {*u*,*v*,*w*} are integer values in the range: $u \in \{0,...,nSlices[0]-1\}$, $v \in \{0,...,nSlices[1]-1\}$, and $w \in \{0,...,nSlices[2]-1\}$. Given *u* the row index, *v* the column index, and *w* the plane index, the ordering of the reference-grid points in the **gridCoord** buffer is row first, column second, and plane third. When the MESHGRID node is initialized from a binary stream, then the contents of **gridCoord** is ignored, and it will be overridden with the coordinates decoded from the binary stream as explained in subclause 5.2.3.2. When the **gridCoord** field is updated by BIFS, the coordinates of the grid points that did not change can be updated in a hierarchical way as explained in subclause 4.3.4.3.1.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **vertexLink** field defines the connectivities between the vertices in the connectivity-wireframe. The decoding of the contents of this field is explained in subclause 5.2.3.1. If the node is initialized from a binary stream, then the content of vertexLink is ignored.

The **vertexOffset** field specifies the scalar offset for each vertex relative to a corresponding reference-grid point. The meaning of the offset is explained in subclause 4.3.4.3.2, while the decoding of this field is described in subclause 5.2.3.3. The value of the scalar offset lies in the range [0, 1). If the node is initialized from a binary stream, then the content of vertexOffset is ignored, and it will be overridden with the scalar offsets decoded from the binary stream.

The **coord** field is filled with the coordinates (*x*,*y*,*z*) of the vertices, as result of the decoding procedure (see subclause 5.2.3.1). Its contents, if any, will always be overridden. The coordinates of the vertices can be derived from the coordinates of the reference-grid points as explained in subclause 5.2.3.1.

The **coordIndex** field is filled with the indices of the triangles when building the triangulation of the decoded mesh as explained in 5.2.3.1.3. Its contents, if any, will always be overridden.

The **nVertices** field stores the number of vertices for each resolution level. This field is updated after the decoding procedure (see subclause 5.2.3.1).

The **hierarchicalLevel** field defines the resolution level of the reference-grid points (**gridCoord** field) that may change during the animation. The usage is explained in subclause 5.2.3.1.

The **filterType** field defines the type of filter used for the hierarchical interpolation of the reference-grid points during the animation.

filterType	Meaning
00	Use short filter (see subclause 5.2.3.1)
01	Use smooth filter (see subclause 5.2.3.1)
10	Reserved
11	Reserved

The **displayLevel** field specifies the resolution level of the MeshGrid model to be rendered. Accepted values are in the range [0, maxLevel], where maxLevel is the maximum value between **nLevels[0], nLevels[1], nLevels[2].** More detail can be found in Annex B.1.1.

The fields **normal**, **normalIndex**, **texCoord**, **texCoordIndex**, **color**, **colorIndex,** and **solid** have the same semantic as for the **IndexedFaceSet** node.

4.3.4.3 Animation Extensions

4.3.4.3.1 Hierarchical interpolation of the reference-grid points

In case of a multi-resolution MeshGrid model, animating the hierarchical reference-grid can be done on a hierarchical basis, more precisely any change in the coordinates of reference-grid points, i.e. the ***gridCoord*** field (see subclauses 4.3.4.2.1 and 4.3.4.2.2), at a certain resolution level will trigger a local update of the coordinates of the reference-grid points belonging to a higher resolution level. The new positions of the reference-grid points belonging to resolution level (*l*+1) can be computed from the new positions of the neighboring reference-grid points at resolution level (*l*) via an interpolation method based on "Dyn's four point scheme for curves" [33]. The position of a reference-grid point P_{2n}^{l+1} (shown in Figure 22) is computed as follows:

$$P_{2n}^{l+1} = \Delta \mathbb{P}_{2n}^{l+1} + \left(-w \times P_{2n-3}^{l} + \left(\tfrac{1}{2} + w\right) \times P_{2n-1}^{l} + \left(\tfrac{1}{2} + w\right) \times P_{2n+1}^{l} - w \times P_{2n+3}^{l}\right),$$

Equation 3

© ISO/IEC 2006 – All rights reserved

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

$$\Delta\mathbb{P}_{2n}^{l+1} = \mathbb{P}_{2n}^{l+1} - \left(-w \times \mathbb{P}_{2n-3}^{l} + \left(\tfrac{1}{2} + w\right) \times \mathbb{P}_{2n-1}^{l} + \left(\tfrac{1}{2} + w\right) \times \mathbb{P}_{2n+1}^{l} - w \times \mathbb{P}_{2n+3}^{l}\right),$$

Equation 4

where *l* represents the hierarchical level of the grid point, $\mathbb{P}$ the last position of a point, *P* the new position, $\Delta\mathbb{P}$ the detail computed for the last position $\mathbb{P}$. The detail $\Delta\mathbb{P}$ is added to the interpolated value in point *P*. The weight *w* defines the smoothness of the limit curve, and can be specified via the ***filterType*** field of the **MeshGrid** node. The weight *w* may have one of the following two values: *w* = 0 when ***filterType*** = 0 and *w* = 1/16 when ***filterType*** = 1. In general *w* is taken equal to 1/16 which corresponds to fitting a Catmull-Rom or Cardinal spline curve through the points.

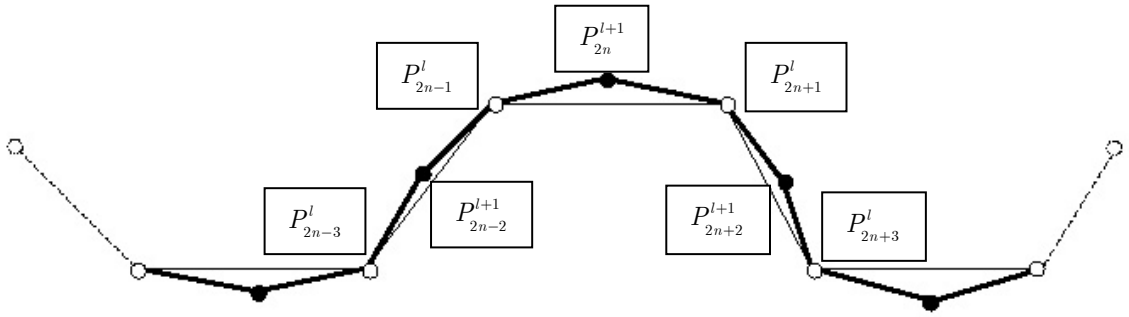


Figure 22 — Dyn's four point scheme for curves applied for the hierarchical MeshGrid.

The ***hierarchicalLevel*** field specifies the resolution level at which the grid points, i.e. the ***gridCoord*** field, may change. The interpolation of the grid points will take place only for the range of resolution levels [***hierarchicalLevel*** + 1, ***nLevels*** – 1]. If the value of ***hierarchicalLevel*** ≥ ***nLevels*** – 1 than no interpolation occurs. In the third case, when ***hierarchicalLevel*** = -1, grid points may change at any resolution level simultaneously, the points that change are detected, and the interpolation of the remaining grid points is done according to Equation 3.

The position of the vertices is updated from the grid positions as explained in subclause 4.3.4.3.2.

4.3.4.3.2 Relationship between the reference-grid points and vertices

The relationship between the vertices of the connectivity-wireframe and reference-grid points is explained in the 2D cross-section of Figure 23, in which the reference-grid is put on top of the object's section.

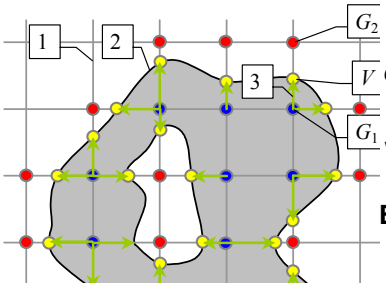
Any reference-grid line *l* (label 1) is the result of the intersection between two reference-surfaces S_1 and S_2, belonging to two different sets (see subclause 4.3.4.1.1). Every vertex *V*, lying on a contour (label 2) of the object, stores the discrete position (*u,v,w*) – which has been derived during decoding – of the grid point it is attached to. The vertices are related to the reference-grid points in two different ways, as follows:

As shown in Figure 23(a)-(b), vertex *V* is positioned in-between two grid points, one inside the object G_1 to which it is attached, and one outside the object G_2. The relative position of vertex *V* with respect to G_1 and G_2 is given by the scalar *offset* (label 3) expressed by Equation 5. Because *offset* indicates the fractional position between G_1 and G_2, any displacement of G_1 or G_2 (see the example of Figure 23 (b)) is automatically reflected in an update of the coordinates (*x,y,z*) of *V* through Equation 6.

As illustrated in Figure 23(c), vertex *V* is attached in this case to a grid position *G* belonging to a *single-layer* reference-grid, i.e. the number of slices is equal to 1 in one of the {*u*,*v*,*w*} directions. The relative position of vertex *V* with respect to *G* is given by the scalar *offset* (label 3) expressed by Equation 7. Similarly to the observation made at the previous point, any displacement of *G* is automatically reflected in an update of the coordinates (*x,y,z*) of *V* through Equation 8. In this case

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

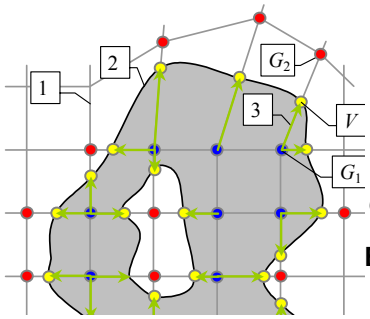
the displacements along the normal follow both directions as shown for the vertices with labels 4 and 5.



$$offset = \frac{|\overrightarrow{G_1V}|}{|\overrightarrow{G_1G_2}|}, \text{ with } offset \in [0,1)$$

Equation 5

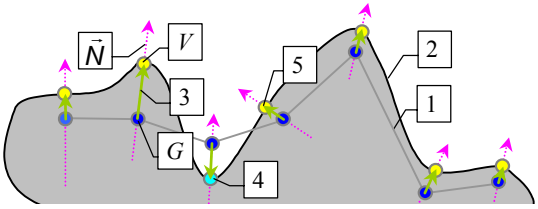
(a)



$$\overrightarrow{OV} = \overrightarrow{OG_1} + \overrightarrow{G_1G_2} \bullet offset$$

Equation 6

(b)



$$offset = \frac{|\overrightarrow{GV}|}{offAmp} + 0.5, \text{ with } offset \in [0,1)$$

Equation 7

where *offAmp* is the maximum amplitude of the offset defined by the *offsetAmplitude* variable specified in MeshGridDecoderConfig (subclause 5.2.2.2).

$$\overrightarrow{OV} = \overrightarrow{OG} + \vec{N} \bullet (offAmp \bullet (offset - 0.5)),$$

Equation 8

where $\vec{N}$ is the normal vector to the surface at vertex *V*.

(c)

In this figure *O* denotes the origin of the reference coordinate system in which the reference-grid is defined.

Figure 23 — A cross-section through a 3D object, illustrating the contour of the object, the reference-grid, and the relation between the vertices (belonging to the connectivity-wireframe and located at the surface of the object) and the grid points.

The offsets of the vertices are stored in the ***vertexOffset*** field of the **MeshGrid** node (see subclause 4.3.4.2).

4.3.4.4 Examples

The following example loads a VRML MeshGrid node.

```
Shape {
   appearance Appearance {
      material Material {
         diffuseColor 1.0 0.0 0.0
         ambientIntensity 0.1
         }
   }
   geometry MeshGrid {
      nSlices [ 3 3 3 ]
      nLevels 1
      displayLevel
     hierarchicalLevel 0
      gridCoord Coordinate {
        point [
         -1.000000 -1.000000 -1.000000,
          0.000000 -1.000000 -1.000000,
```

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          ...
          0.000000  1.000000  1.000000,
          1.000000  1.000000  1.000000
        ]
      }
      vertexLink [
        1 1 1, 32
        2 2
        ...
        2 2
        2 2
      ]
      vertexOffset [
        0.500000
        0.200000
        ...
        0.200000
        0.500000
      ]
    }
}
```

4.3.5 MorphSpace

4.3.5.1 Introduction

Morphing is mainly an interpolation technique used to create from two objects a series of intermediate objects that change continuously, in order to make a smooth transition from the source to the target. A straight extension of the morphing between two elements –the source and the target- consists in considering a collection of possible targets and compose a virtual object configuration by weighting those targets. This collection represents a basis of animation space and animation is performed by simply updating the weight vector. The following node allows the representation of a mesh as a combination of a base shape and a collection of target geometries.

4.3.5.2 MorphShape node

4.3.5.2.1 Synthax

MorphShape{ #%NDT=SF3DNode,SF2DNode
exposedField SFInt32 morphID
exposedField SFShapeNode baseShape
exposedField MFShapeNode targetShapes [ ]
exposedField MFFloat weights [ ]
}

4.3.5.2.2 Semantics

morphID - a unique identifier between 0 and 1023 which allows that the morph to be addressed at animation run-time.

baseShape – a Shape node that represent the base mesh. The geometry field of the baseShape can be any geometry supported by ISOIEC 14496 (e.g. IndexedFaceSet, IndexedLineSet, SolidRep).

targetShapes – a vector of Shapes nodes representing the shape of the target meshes. The tool used for definig an appearance and a geometry of a target shape must be the same as the tool used for defining the appearance and the geometry of the base shape (e.g. if the baseShape is defined by using IndexedFaceSet, all the target shapes must be defined by using IndexedFaceSet).

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

weights – a vector of integers of the same size as the **targetShapes**. The morphed shape is obtained according to the following formula:

$$M = B + \sum_{i=1}^{n} (T_i - B) * w_i$$

Equation 9 — Morphing computation.

with

M –morphed shape,

B – base shape,

T_i – target shape *i*,

W_i – weight of the T_i.

The morphing is performed for all the components of the Shape (Appearance and Geometry) that have different values in the base shape and the target shapes (e.g. if the base shape and the target shapes are definined by using IndexedFaceSet and the *coord* field contains different values in the base shape and in the target geometries, the *coord* component of the morph shape is obtained by using Equation 9) applied to the *coord* field. Note that the size of the *coord* field must be the same for the base shapes and the target shapes).

If the shapes (base and targets) are defined by using IndexedFaceSet, a tipical decoder should support morphing of the following geometry components: *coord*, *normals*, *color*, *texCoord*.

4.4 Solid representation

Solid representation includes 3 geometry nodes (Implicit, Quadric and SolidRep) and extensions of the script language for implementation of the Arithmetic of Forms.

4.4.1 Solid modeling nodes

4.4.1.1 Implicit

4.4.1.1.1 Node interface

```
Implicit { # % NDT = SFGeometryNode
    exposedField  SFVec3f    bboxSize   2.0 2.0 2.0
    exposedField  MFFloat    c          [ ]  # 4 coeffs for hyperplane
                                              #10 coeffs for quadrics,
            # 35 coeffs for quartics
    exposedField  SFBool     solid      FALSE
    exposedField  SFBool     dual       FALSE
    exposedField  MFInt32    densities  []   # ( 1, 2)
}
```

© ISO/IEC 2006 – All rights reserved

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

4.4.1.1.2 Functionality and semantics

The **Implicit** geometry node defines an algebraic surface represented by an implicit equation, which makes it possible to know if an unspecified point of space is inside, on or outside the volume delimited by this surface.

bboxSize : bounding box dimension

c : coefficients of following polynomials as follows:

The implicit equation that defines an hyperplane, is as follows:

$$c_0 X_0 + c_1 X_1 + c_2 X_2 + c_3 X_3 = 0$$

The implicit equation of the second degree that defines quadrics, is as follows:

$$c_0 X_0^2 + c_1 X_0 X_1 + c_2 X_1^2 + c_3 X_0 X_2 + c_4 X_1 X_2 + c_5 X_2^2 + c_6 X_0 X_3 + c_7 X_1 X_3 + c_8 X_2 X_3 + c_9 X_3^2 = 0$$

The implicit equation of the fourth degree that defines quartics is as follows:

$$c_0 X_0^4 + c_1 X_0^3 X_1 + \ldots + c_4 X_1^4 + c_5 X_0^3 X_2 + \ldots + c_{14} X_2^4 + c_{15} X_0^3 X_3 + \ldots + c_{34} X_3^4 = 0$$

In the above equations, the point coordinates were made homogeneous by the addition of a fourth coordinate X_3.

solid indicates if the geometry is solid (TRUE) or is a surface (FALSE)

dual indicates that a unary duality operation has to be applied to volume initialization. Implies **solid** == TRUE.

densities has 2 values: the first value is the density code associated to the skin and the second value is the density associated to the inside.

The surface is only created when the node is displayed allowing its inclusion into a **SolidRep** node. The origin of the axis system is the bounding box's center. The surface will be clipped by the bounding box. If **solid** is set to TRUE, the volume will be closed and intersected with the bounding box.

If the **dual** field is TRUE, the volume is set to **solid** and the difference operation with bounding box will be carried out (bounding box – implicit solid).

The use of bounding boxes for **Implicit** nodes makes it possible to limit infinite surfaces.

The densities are used when including the primitive into a **SolidRep** node with solid operation. The density value is an Int32 and can be encoded.

As a benefit of the implicit definition, the function *isOutside (Point &p)* will return 3 values:

1 : if the point is outside the volume defined by the surface

0 : if the point is on the surface

-1 : if the point is inside the volume defined by the surface

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.4.1.2 Quadric

4.4.1.2.1 Node interface

```
Quadric { #%NDT=SFGeometryNode
    exposedField  SFVec3f    bboxSize    2 2 2
    exposedField  SFVec4f    P0          -1 0 0 1
    exposedField  SFVec4f    P1          1 0 0 1
    exposedField  SFVec4f    P2          0 1 0 0
    exposedField  SFVec4f    P3          0 0 1 0
    exposedField  SFVec4f    P4          0 1 0 1
    exposedField  SFVec4f    P5          0 0 1 1
    exposedField  SFBool     solid       FALSE
    exposedField  SFBool     dual        FALSE
    exposedField  MFInt32    densities   [] # ( 1, 2)
}
```

4.4.1.2.2 Functionality and semantics

The **Quadric** geometry node defines a second-degree implicit surface by using 6 geometric control points.

bboxSize bounding box dimension

The 6 geometric control points of the quadric, in projective coordinates (see 4.4.1.2.2.2) are:

P0, **P1** 2 points tangent to the quadric

P2, **P3** 2 poles of the construction tetrahedron

P4, **P5** 2 passing points of the quadric

solid solid (TRUE) or surface (FALSE)

dual unary duality operation to be applied to volume initialization. Implies solid == TRUE.

densities 2 values: the first value is the density code associated to the skin and the second value is the density associated to the inside.

Each point is defined using homogeneous coordinates allowing the point to be sent to the infinity. The values are relative to the unitary bounding box (from –1 to +1). If the absolute value of a coordinate is greater than 1, the point is outside the bounding box.

This node creates an **Implicit** node of the second degree through a geometric interface. The polynomial coefficients will be calculated according to the geometrical construction method using a construction tetrahedron and 2 passing points as described in the Quadric's construction mechanism (see subclause 4.4.1.2.2.1)**.**

The quadric surface is only created when the node is displayed allowing its inclusion into a **SolidRep** node. The origin of the axis system is the bounding box's center. The surface will be clipped by the bounding boxe. If **solid** is set to TRUE, the volume will be closed and intersected with the bounding box.

If the **dual** field is TRUE, the volume is set to **solid** and the difference operation with the bounding box will be carried out (bounding box – implicit solid).

A continuous volume deformation is implemented by moving the control points.

The densities are used when including the primitive into a **SolidRep** node with solid operation (see subclause 4.4.1.3). The density value is encoded as a 32-bit integer.

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

© ISO/IEC 2006 – All rights reserved

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

4.4.1.2.2.1 Quadric's Construction Mechanism

Based on the principle of projective geometry, a geometrical construction mechanism for quadrics has been defined in [66], [68], [55] according to Pascal's Theorem.

In fact, the geometric control of any quadric goes via the extension to 3D of the principle for constructing conics, using a construction triangle and a passing point. The quadric will be constructed and controlled with the help of a "construction tetrahedron" and two passing points located in the planes defined by two particular faces of the tetrahedron. A set of six points will thus allow us to define the quadric from two conic sections that intersect in space at the two contact points on the tangent planes.

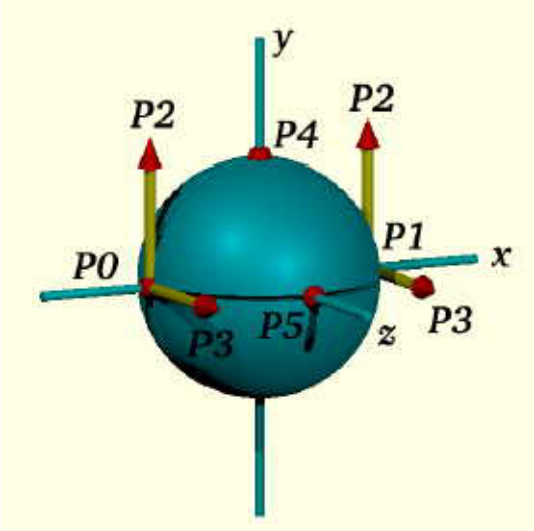


Figure 24 — Quadric's 6 geometric control points

The sphere in the above figure is constructed with the help of two circular sections belonging to two perpendicular planes. The two points P2 and P3 that serve as poles have been sent to infinity respectively along the Y-axis and the Z-axis (fourth coordinate equal to 0).

In general, for any quadric - projective, affine or metric - we shall define just six control points, P0 to P5, such that P0 and P1 are two contact points on two tangent planes to the quadric, P2 and P3 are two poles of the quadric and therefore belong to the two tangent planes, and P4 and P5 are two passing points for the quadric defining two conic sections of the quadric in the planes P0-P1-P2 and P0-P1-P3.

It is possible to determine the 10 coefficients of the quadric's general equation from the previously described quadric's geometrical construction system. (Reference: [66] – pages 100 – 114).

4.4.1.2.2.2 Projective Coordinates System

In order to define all the quadrics with the 6 control points, it is imperative to remain in projective space. The coordinates system used in projective space is that of Grassmann-Plücker (one refers then to homogeneous coordinates), in which a point, a line and a plane respectively have 4, 6 and 4 coordinates. It can be interesting to make a parallel with non-homogeneous Euclidean coordinates. In this respect, one uses a representation of the point in 3 dimensions of the form $\psi\ \psi\psi\psi\psi$ -. Adding to this triplet a fourth coordinate equal to 1 does not modify in any way the correct operation of all geometrical procedures known as Euclidean. The first 3 coordinates will indifferently represent integers or decimal numbers, provided that the fourth coordinate remains equal to 1. By particularizing the fourth coordinate and giving it a 0 value, one goes from a Euclidean coordinates system to an affine coordinates system, in which it becomes possible to control and use the points to infinity. Thus, an observer located in (1 1 1 0) will be positioned at infinity in the direction determined by its Euclidean position (1 1 1 1) and the origin of the coordinates system (0 0 0 1). In addition to the possibility of managing infinity in a natural way, this approach makes it possible to solve the problem of representation and calculation of rational coordinates, which is fundamental to obtain perfect precision

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

whenever the coordinates are initially rational or brought about to become rational. Let us consider for example the homogeneous coordinates point (1/3 5/6 0 1); it will be represented by the approximate quadruplet (0.333 0.833 0 1) or, much better, by the quadruplet (2/6 5/6 0/6 6/6) – – (2: 5: 0: 6).

4.4.1.3 SolidRep

4.4.1.3.1 Node interface

```
SolidRep { #%NDT=SFGeometryNode
    exposedField  SFVec3f    bboxSize   2.0 2.0 2.0
    exposedField  SF3DNode   solidTree  NULL
    exposedField  MFInt32    densityList []
}
```

4.4.1.3.2 Functionality and semantics

The **SolidRep** geometry node holds the solid tree resulting of solid operations on solid primitives and/or others **SolidRep** nodes, and/or other BIFS 3D geometrical nodes.

bboxSize contains the bounding box dimensions that will be used for clipping.

solidTree contains the geometry to be initialized as solid. The following nodes may be included under a **solidTree** field: **Group**, **Transform**, **Shape**, **Implicit**, **Quadric**, **SolidRep**, **Sphere**, **Cone**, **Cylinder**, **IndexedFaceSet**, and **Box**.

The **solidTree** field can also be modified by solid operations applied by a script.

densityList: set of densities to select for display . If the list is empty, the default value will be **all densities.**

A **SolidRep** node can be included in a **Transform** node. The implicit operation of the children of a **Transform** node or of any grouping node is the ternary union operation.

Texture mapping onto implicit surfaces can be done using any algorithm. The gradient algorithm [93] is an example of an algorithm well suited for implicit surfaces.

4.4.2 Scripting extensions for solid modeling operations

The result of a solid operation is always preserved in the **solidTree** field of a **SolidRep** node. The **SolidRep** node is always the root of a solid tree. This solid tree is not processed until the display request. Only the root of the complete tree will be processed (and not the sub-trees).

These operations are accessed through script language, whose selected operators (+, -...) were overloaded for operations between **Geometry** nodes.

A non-implicit solid geometrical node is first of all transformed into a **SolidRep** node before being used as a solid operation's operand. If this geometry does not define a closed volume, it will not be considered.

The syntax of the solid operation is as follows:

SolidRep.solidTree = {Geometry / Int32} < op > {SolidRep/Implicit/Quadric}

The operands are:

— An implicit primitive (**Implicit** or **Quadric**) with "**solid** = TRUE" and densities;

— Or a **SolidRep** containing a sub-solid tree;

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

© ISO/IEC 2006 – All rights reserved

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

— Or a non-implicit geometry (only solid primitives are supported such as **Sphere**, **Cylinder**, **Cone**, **Box** or closed shapes defined by **IndexedFaceSet**);

— Or an 32-bit integer (**Int32**) value representing the density of the entire space (used as operand for implicit filtering of densities)

An only-assignment expression makes it possible to convert a non-implicit geometrical primitive into a bound implicit solid.

The complete set of solid operations <**op**> is composed of general arithmetic operators, logic operators (ternary logic) and filtering operators. A left to right evaluation of the expression is applied. The parenthesis can be used for grouping with precedence on the left to right evaluation.

4.4.2.1 General arithmetic operators

The operators below achieve combinations of solid forms by isolating or selecting certain regions in a combinatorial way by means of their densities. The densities are always positive integer.

Table 2 — General arithmetic operators.

Op	Description	Rules for each P(X1,X2,X3,X4)	Syntax
Sadd	Arithmetic addition of the density of two forms	$d_r(P) = d_0(P) + d_1(P)$	**Sadd (F0, F1)**
Smul	Arithmetic multiplication of the density of two forms	$d_r(P) = d_0(P) * d_1(P)$	**Smul (F0, F1)**
Sdif	The positive difference of two forms *F0* and *F1*.	Returns the difference between the densities if this result is nonnegative, and 0 if the result is negative.	**Sdif (F0, F1)**
Sexp	Exponentiation of forms	Raises one form to the power of another form. The density of a point with respect to *F1* serves as exponent to the density of the same point with respect to *F0*.	**Sexp (F0, F1)**
Sgcd	Greatest common divisor	Returns the *gcd* of the densities of two forms.	**Sgcd (F0, F1)**
Slcm	Least common multiple	Returns the *lcm* of the densities of two forms.	**Slcm (F0, F1)**
Smod	Integral remainder	The Integral remainder operator calculates the integer remainder when the density of a point with respect to *F0* is divided by the density of that point with respect to *F1*	**Smod (F0, F1)**
Ssab	Absolute difference	Returns the result of the subtraction between the densities of two forms when this result is nonnegative. Otherwise, the return is the result opposed value.	**Ssab (F0, F1)**
Scub	Integral cube root	Returns the integral cube root of the density of the form.	**Scub (F0)**
Ssqr	Integral square root	Returns the integral square root of the density of the form.	**Ssqr (F0)**
Smax	Maximum	This operator is equivalent to the ternary union for n-ary logic	**Smax (F0, F1)**
Smin	Minimum	This operator is equivalent to the ternary intersection for n-ary logic.	**Smin (F0, F1)**

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.4.2.2 Logic operators (Ternary logic)

The forms are coded with ternary logic: the density 0 for the outside, the density 1 for the skin and the density 2 for the inside.

Note: if used on an n-ary logic, the densities are taken modulo 2 to convert the value into the ternary logic.

Table 3 — Ternary logic operators.

<table>
<tr><th>Op</th><th>Description</th><th>Rules for each P(X1,X2,X3,X4)</th><th>Syntax</th></tr>
<tr><td>Suni</td><td>Ternary union of two forms.</td><td>
<table>
<tr><td></td><td></td><td colspan="3">F1</td></tr>
<tr><td></td><td>Suni</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td></td><td>0</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>F0</td><td>1</td><td>1</td><td>1</td><td>2</td></tr>
<tr><td></td><td>2</td><td>2</td><td>2</td><td>2</td></tr>
</table>
</td><td>Suni (F0, F1)</td></tr>
<tr><td>Sint</td><td>Ternary intersection</td><td>
<table>
<tr><td></td><td></td><td colspan="3">F1</td></tr>
<tr><td></td><td>Sint</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td></td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>F0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td></td><td>2</td><td>0</td><td>1</td><td>2</td></tr>
</table>
</td><td>Sint (F0, F1)</td></tr>
<tr><td>Simp</td><td>Ternary implication</td><td>
<table>
<tr><td></td><td></td><td colspan="3">F1</td></tr>
<tr><td></td><td>Simp</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td></td><td>0</td><td>2</td><td>2</td><td>2</td></tr>
<tr><td>F0</td><td>1</td><td>1</td><td>1</td><td>2</td></tr>
<tr><td></td><td>2</td><td>0</td><td>1</td><td>2</td></tr>
</table>
</td><td>Simp (F0, F1)</td></tr>
<tr><td>Simr</td><td>Reciprocal Ternary implication</td><td>
<table>
<tr><td></td><td></td><td colspan="3">F1</td></tr>
<tr><td></td><td>Simr</td><td>0</td><td>1</td><td>2</td></tr>
<tr><td></td><td>0</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>F0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td>2</td><td>0</td><td>1</td><td>2</td></tr>
</table>
</td><td>Simr (F0, F1)</td></tr>
<tr><td>Sdua</td><td>Ternary dual of the volume</td><td>
<table>
<tr><td></td><td>Sdua</td><td></td></tr>
<tr><td></td><td>0</td><td>2</td></tr>
<tr><td>F0</td><td>1</td><td>1</td></tr>
<tr><td></td><td>2</td><td>0</td></tr>
</table>
</td><td>Sdua (F0)</td></tr>
</table>

4.4.2.3 Filtering/test operators

In addition a set of test functions can be applied on ***SolidRep*** nodes. These functions are used to filter densities while keeping the filtering inside the solid tree instead of defining the densities to be considered during display time.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 4 — Filtering and test operators.

Filter	Description	Rules	Syntax
Seqf	Equality filter	The density of the volume that checks the equality test, and 0 otherwise.	**Seqf (F0, F1)**
Sgtef	greater than or equal filter	The density of the second volume if the density of the first one is greater than or equal to the density of the second volume, and 0 otherwise.	**Sgtef (F0, F1)**
Sgtf	Greater than filter	The density of the second volume if the density of the first one is greater than the density of the second volume, and 0 otherwise.	**Sgtf (F0, F1)**
Sltef	Less than or equal filter	The density of the second volume if the density of the first one is less than or equal to the density of the second volume, and 0 otherwise.	**Sltef (F0, F1)**
Sltf	Less than filter	The density of the second volume if the density of the first one is less than the density of the second volume, and 0 otherwise.	**Sltf (F0, F1)**
Sevnf	Even filter	The volume density if the density is even, and 0 otherwise	**Sevnf (F0)**
Soddf	Odd filter	The result is the volume density if the density is odd, and 0 otherwise.	**Soddf (F0)**
Sneqf	Difference filter	The result is the second volume density if the densities are different, and 0 otherwise.	**Sneqf (F0, F1)**

Scripts are used through **Script** node. A **Script** node can be included as descending from any grouping node but is independent of the current coordinate system.

4.4.3 Examples

A cell primitive

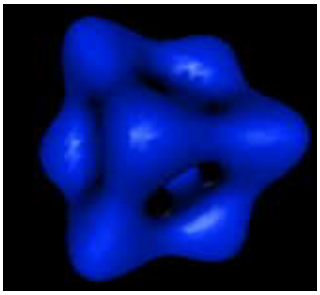
Dual of quarter cylinder defined by projective hexahedron



```
# Dual of quarter cylinder
PROTO GDlychxa [
   fieldSFVec3f size 5 5 5
   field SFBool dual TRUE
   ]{
      Quadric {
   P0 -3 0 0 1
   P1  1 0 -1 1
   P2  0 1 0 0
   P3  0 0 1 0
   P4  0 1 0 0
   P5  -1 0 1 1
            bboxSize IS size
   dual IS dual
      }
}
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A quartic defined by 35 coefficients



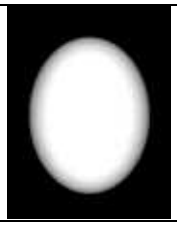
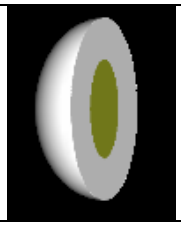
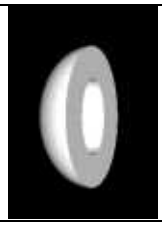
Tangle cube

VRML code:

```
Group {
   children Shape {
# tangle cube
      geometry Implicit {
         bboxSize 5 5 5
         solid FALSE
         c [
            -1,0,0,0,-1,0,0,0,0,0,0,0,0,0,-1,0,
            0,0,0,0,0,0,0,0,5,0,5,0,0,5,0,
            0,0,-10.2]
      }
   }
```

Solid operations and densities

Example of density use: a single model carries the characteristic of the matter and can be displayed differently

Egg white : density 1 **Egg yolk :** density 3 **Operation:** addition					
	Full egg with yolk inside	Cut by a box (density 4)	Cut by a box (density 1)	Cut by a box (density 4)	Cut by a box (density 4)
Densities displayed	1,3,4	1,4	1,3,4	1	4

VRML code:

```
# b: egg cut by a box
Group{
    children [
        # Primary SolidRep
        DEF Le_Solid3 SolidRep{
            bboxSize 5 5 5
            densityList [1 3 4] # choice of densities to display
            dual FALSE
        }
        DEF Le_Script Script{ # Solid Tree definition
            field SFNode Srepout USE Le_Solid3
            #primitives
            field SFNode White # container
```

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                Transform {
                translation 0 0 0
                    children [
                        Shape {
                        …
                        geometry
                            Quadric { P0 1 0 0 1
                            P1 -1 0 0 1
                            P2 0 1 0 0
                            P3 0 0 1 0
                            P4 0 1 0 1
                            P5 0 0 1 1
                            boxSize 1 1.3 0.95
                            solid FALSE}
                        }
                    ]
                }
            field SFNode Yolk # matter inside
                Transform {
                    children [
                        Shape {
                        …
                        geometry
                            Quadric { P0 1 0 0 1
                            P1 -1 0 0 1
                            P2 0 1 0 0
                            P3 0 0 1 0
                            P4 0 1 0 1
                            P5 0 0 1 1
                            bboxSize 0.7 0.7 0.4
                            density 3
                            solid FALSE}
                        }
                    ]
                }
            field SFNode CuttingBox # cutting tool
                Transform {
                translation 0.5 0 0
                    children [
                        Quadric { P0 -1 0 0 1
                        P1 1 0 0 1
                        P2 0 1 0 0
                        P3 0 0 1 0
                        P4 0 1 0 0
                        P5 0 0 1 0
                        bboxSize 1 1 1
                        density 4
                        solid FALSE}
                    ]
                }
            directOutput TRUE
            url "vrmlscript:
                function initialize(){
                    Srepout.solidTree = Sdif(Sadd(White,Yolk), CuttingBox);}
            "
        }
    ]
}
```

4.5 Texture tools

4.5.1 Introduction

The technologie defined in this subclause is called "Depth image-based representation" and comprises the following nodes: **DepthImage**, **SimpleTexture**, **PointTexture**, and **OctreeImage**.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2006 – All rights reserved
Not for Resale

4.5.2 Depth Image-Based Representation

4.5.2.1 DepthImage

4.5.2.1.1 Node interface

```
DepthImage { #%NDT=SF3DNode
    exposedField    SFVec3f          position      0 0 10
    exposedField    SFRotation       orientation   0 0 1 0
    exposedField    SFVec2f          fieldOfView   π/4 π/4
    exposedField    SFFloat          nearPlane     10
    exposedField    SFFloat          farPlane      100
    exposedField    SFBool           orthographic  TRUE
    field           SFDepthTextureNode diTexture   NULL
}
```

4.5.2.1.2 Functionality and semantics

The **DepthImage** node defines a single IBR texture. When multiple **DepthImage** nodes are related to each other, they are processed as a group, and thus, should be placed under the same Transform node.

The **diTexture** field specifies the texture with depth, which shall be mapped into the region defined in the **DepthImage** node. It shall be one of the various types of depth image texture (**SimpleTexture** or **PointTexture**).

The **position** and **orientation** fields specify the relative location of the viewpoint of the IBR texture in the local coordinate system. **position** is relative to the coordinate system's origin (0, 0, 0), while **orientation** specifies a rotation relative to the default orientation. In the default position and orientation, the viewer is on the Z-axis looking down the –Z-axis toward the origin with +X to the right and +Y straight up. However, the transformation hierarchy affects the final position and orientation of the viewpoint.

The **fieldOfView** field specifies a viewing angle from the camera viewpoint defined by **position** and **orientation** fields. The first value denotes the angle to the horizontal side and the second value denotes the angle to the vertical side. The default values are 45 degrees in radiant. However, when **orthographic** field is set to TRUE, the **fieldOfView** field denotes the width and height of the near plane and far plane.

The **nearPlane** and **farPlane** fields specify the distances from the viewpoint to the near plane and far plane of the visibility area. The texture and depth data shows the area closed by the near plane, far plane and the **fieldOfView**. The depth data are scaled to the distance from **nearPlane** to **farPlane**.

The **orthographic** field specifies the view type of the IBR texture. When set to TRUE, the IBR texture is based on orthographic view. Otherwise, the IBR texture is based on perspective view.

The **position, orientation, fieldOfView, nearPlane, farPlane,** and **orthographic** fields are exposedField types, which are for extrinsic parameters. The DepthImage node supports the camera movement and the changeable view frustum corresponding to movement or deformation of a DIBR object. And reference images that are suitable to the characteristic of a DIBR model are obtained in the modeling stage. Therefore, the fields that reflect the camera movement and the the changeable view frustum and the reference images in the modeling stage are used to create a view frustum and a DIBR object in the rendering stage.

© ISO/IEC 2006 – All rights reserved

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

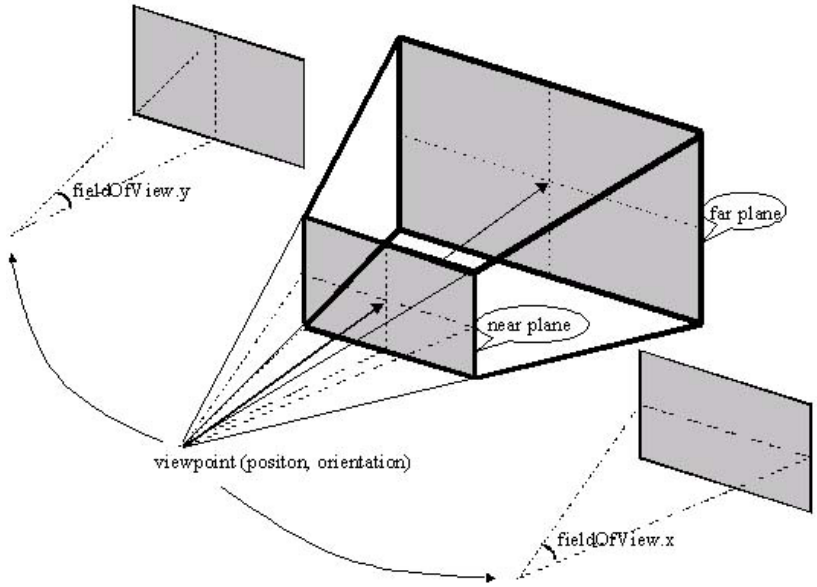


Figure 25 — Perspective view of the DepthImage

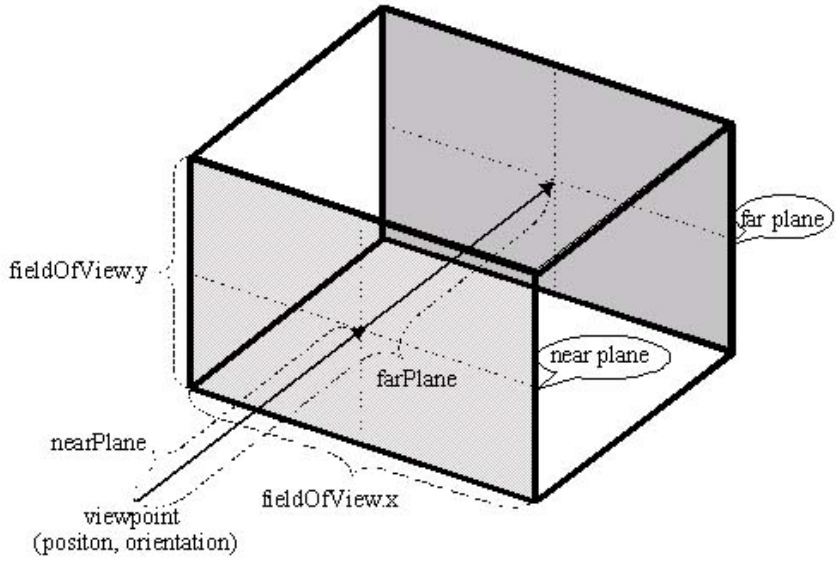


Figure 26 — Orthographic view of the DepthImage

4.5.2.2 SimpleTexture

4.5.2.2.1 Node interface

```
SimpleTexture { #%NDT=SFDepthTextureNode
    field    SFTextureNode    texture     NULL
    field    SFTextureNode    depth       NULL
}
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.2.2.2 Functionality and semantics

The **SimpleTexture** node defines a single layer of IBR texture.

The **texture** field specifies the flat image that contains color for each pixel. It shall be one of the various types of texture nodes (**ImageTexture**, **MovieTexture** or **PixelTexture**).

The **depth** field specifies the depth for each pixel in the **texture** field. The size of the depth map shall be the same size as the image or movie in the **texture** field. Depth field shall be one of the various types of texture nodes (ImageTexture, MovieTexture or PixelTexture), where only the nodes representing gray scale images are allowed. If the depth field is unspecified, the alpha channel in the texture field shall be used as the depth map. If the depth map is not specified through depth field or alpha channel, the result is undefined.

Depth field allows to compute the actual distance of the 3D points of the model to the plane which passes through the

viewpoint and parallel to the near plane and far plane:

$$dist = nearPlane + (1 - \frac{d-1}{d_{\max}-1})(farPlane - nearPlane),$$

Equation 10

where *d* is depth value and $d_{\max}$ is maximum allowed depth value. It is assumed that for the points of the model, *d > 0*, where *d=1* corresponds to far plane, *d=* $d_{\max}$ corresponds to near plane.

This formula is valid for both perspective and orthographic case, since *d* is distance between the point and the plane. $d_{\max}$ is the largest *d* value that can be represented by the bits used for each pixel:

(1) If the depth is specified through **depth** field, then depth value *d* equals to the gray scale.

(2) If the depth is specified through alpha channel in the image defined via **texture** field, then the depth value *d* is equal to alpha channel value.

The depth value is also used to indicate which points belong to the model: only the point for which d is nonzero belong to the model.

For animated DepthImage-based model, only DepthImage with SimpleTextures as diTextures are used.

Each of the Simple Textures can be animated in one of the following ways:

(1) **depth** field is still image satisfying the above condition, **texture** field is arbitrary MovieTexture

(2) **depth** field is arbitrary MovieTexture satisfying the above condition on the **depth** field, **texture** field is still image

(3) both **depth** and **texture** are MovieTextures, and **depth** field satisfies the above condition

(4) **depth** field is not used, and the depth information is retrieved from the alpha channel of the MovieTexture that animates the **texture** field

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.2.3 PointTexture

4.5.2.3.1 Node interface

```
PointTexture { #%NDT=SFDepthTextureNode
    field    SFInt32        width        256
    field    SFInt32        height       256
    field    MFInt32        depth        []
    field    MFColor        color        []
    field    SFInt32        depthNbBits   7
}
```

4.5.2.3.2 Functionality and semantics

The **PointTexture** node defines a multiple layers of IBR points.

The **width** and **height** field specify the width and height of the texture.

Geometrical meaning of the depth values, and all the conventions on their interpretation adopted for the SimpleTexture, apply here as well.

The **depth** field specifies a multiple depths of each point in the projection plane, which is assumed to be farPlane (see above) in the order of traversal, which starts from the point in the lower left corner and traverses to the right to finish the horizontal line before moving to the upper line. For each point, the number of depths (pixels) is first stored and that number of depth values shall follow.

The **color** field specifies color of current pixel. The order shall be the same as the **depth** field except that number of depths (pixels) for each point is not included.

The **depthNbBits** field specifies the number of bits used for the original depth data. The value of depthNbBits ranges from 0 to 31, and the actual number of bits used in the original data is depthNbBits+1. The d_{max} used in the distance equation is derived as follows:

$$d_{\max} = 2^{(depthNbBits+1)} - 1$$

Equation 11

4.5.2.4 OctreeImage

4.5.2.4.1 Node interface

```
OctreeImage { #%NDT=SF3DNode
    field       SFInt32              octreeResolution  256 #%b=[1,+I]
    field       MFInt32              octree            []  #%b=[0,255] #%q=13 8
    field       MFInt32              voxelImageIndex   []   #%q=13,8
    field       MFDepthImageNode     images            []
}
```

4.5.2.4.2 Functionality and semantics

The **OctreeImage** node defines a TBVO structure, in which an octree structure, corresponding image index array, and a set of **images** exist.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **images** field specifies a set of **DepthImage** nodes with **SimpleTexture** for **diTexture** field; **depth** field in these **SimpleTexture** nodes is not used. The **orthographic** field must be TRUE for the **DepthImage** nodes. For each of **SimpleTexture**, **texture** field stores the color information of the object, or part of the object view (for example, its cross-section by a camera plane) as obtained by the orthographic camera whose position and orientation are specified in the corresponding fields of **DepthImage**. Parts of the object corresponding to each camera are assigned at the stage of model construction. The object partitioning, using the values of **position**, **orientation**, and **texture** fields, is performed so as to minimize the number of cameras (or, equivalently, of the involved **images**), at the same time to include all the object parts potentially visible from an arbitrary chosen position. The **orientation** fields must satisfy the condition: camera view vector has only one nonzero component (i.e., is perpendicular to one of the enclosing cube faces). Also, sides of the **SimpleTexture** image must be parallel to corresponding sides of enclosing cube.

The **octree** field completely describes object geometry. Geometry is represented as a set of voxels that constitutes the given object. An octree is a tree-like data structure, in which each voxel is represented by a byte. 1 in *i*th bit of this byte means that the children voxels exist for the *i*th child of that internal voxel; while 0 means that it does not. The order of the octree internal voxels shall be the order of breadth first traversal of the octree. The order of eight children of an internal voxel is shown in Figure 27. The size of the enclosing cube of the total octree is 1×1×1, and the center of the octree cube shall be the origin (0, 0, 0) of the local coordinate system.

The **voxelImageIndex** field contains an array of image indices assigned to voxel. At the rendering stage, color attributed to an octree leaf is determined by orthographically projecting the leaf onto one of the **images** with a particular index. The indices are stored in an octree-like fashion: if a particular image can be used for all the leaves contained in a specific voxel, the voxel containing index of the image is issued into the stream; otherwise, the voxel containing a fixed 'further subdivision' code is issued, which means that image index will be specified separately for each children of the current voxel (in the same recursive fashion). If the **voxelImageIndex** is empty, then the image indices are determined during rendering stage.

The **octreeResolution** field specifies maximum allowable number of octree leaves along a side of the enclosing cube. The level of the octree can be determined from **octreeResolution** using the following equation:

$$octreeLevel = \lceil \log_2(octreeResolution) \rceil$$

Equation 12

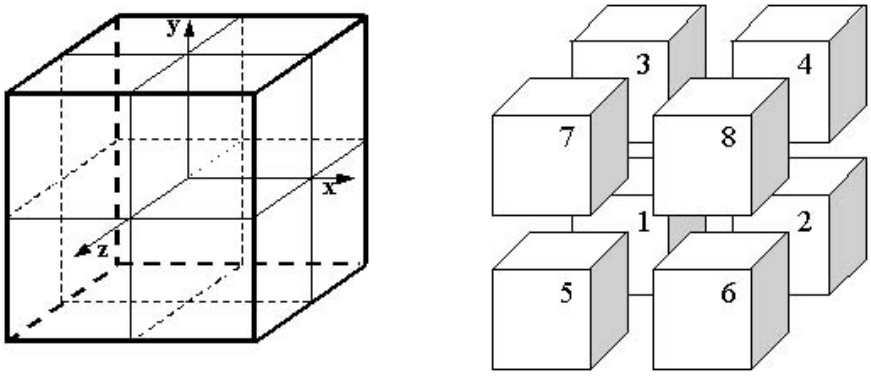


Figure 27 — The structure of octree and the order of the children

Animation of the **OctreeImage** can be performed by the same approach as the first three ways of depth-image-based animation described above, with the only difference of using octree field instead of the depth field.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.3 Depth Image-based Representation Version 2

4.5.3.1 Introduction

Version 1 of DIBR introduced depth image-based representations (DIBR) of still and animated 3D objects. Instead of a complex polygonal mesh, which is hard to construct and handle for realistic models, image- or point-based methods represent a 3D object (scene) as a set of reference images completely covering its visible surface. This data is usually accompanied by some kind of information about the object geometry. To that end, each reference image comes with a corresponding depth map, an array of distances from the pixels in the image plane to the object surface. Rendering is achieved by either forward warping or splat rendering. But with Version 1 of the specification of DIBR nodes no high-quality rendering can be achieved.

Version 2 nodes allow for high-quality rendering of depth image-based representations. High-quality rendering is based on the notion of point-sampled surfaces as non-uniformly sampled signals. Point-sampled surfaces can be easily constructed from the DIBR nodes by projecting the pixels with depth into 3D-space. The discrete signals are rendered by reconstructing and band-limiting a continuous signal in image space using so called resampling filters.

A point-based surface consists of a set of non-uniformly distributed samples of a surface; hence we interpret it as a non-uniformly sampled signal. To continuously reconstruct this signal, we have to associate a 2D reconstruction kernel $r_k(u)$ with each sample point p_k. The kernels are defined in a local tangent frame with coordinates $u = (u, v)$ at the point p_k, as illustrated on the left in Figure 28 . The tangent frame is defined by the splat and normal extensions of the DIBR structures Version 2 [97].

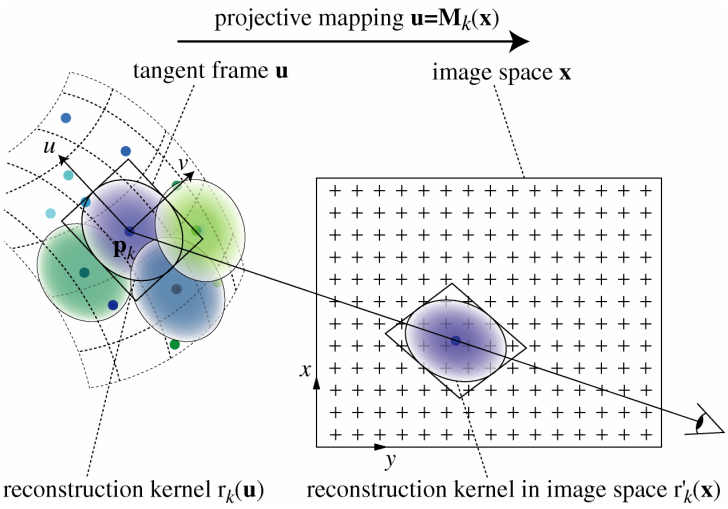


Figure 28 — Local tangent planes and reconstruction kernels

4.5.3.2 DepthImageV2 Node

4.5.3.2.1 Node interface

DepthImageV2 { #%NDT=SF3DNode			
exposedField	**SFVec3f**	**position**	**0 0 10**
exposedField	**SFRotation**	**orientation**	**0 0 1 0**
exposedField	**SFVec2f**	**fieldOfView**	**π/4 π/4**
exposedField	**SFFloat**	**nearPlane**	**10**
exposedField	**SFFloat**	**farPlane**	**100**
field	**SFVec2f**	**splatMinMax**	**0.1115 0.9875**
exposedField	**SFBool**	**orthographic**	**TRUE**
field	**SFDepthTextureNode**	**diTexture**	**NULL**
}			

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.5.3.2.2 Functionality and semantics

The **DepthImageV2** node defines a single IBR texture. When multiple **DepthImage** nodes are related to each other, they are processed as a group, and thus, should be placed under the same Transform node.

The **diTexture** field specifies the texture with depth, which shall be mapped into the region defined in the **DepthImageV2** node. It shall be one of the various types of depth image texture (**SimpleTextureV2** or **PointTextureV2**).

The **position** and **orientation** fields specify the relative location of the viewpoint of the IBR texture in the local coordinate system. **position** is relative to the coordinate system's origin (0, 0, 0), while **orientation** specifies a rotation relative to the default orientation. In the default position and orientation, the viewer is on the Z-axis looking down the –Z-axis toward the origin with +X to the right and +Y straight up. However, the transformation hierarchy affects the final position and orientation of the viewpoint.

The **fieldOfView** field specifies a viewing angle from the camera viewpoint defined by **position** and **orientation** fields. The first value denotes the angle to the horizontal side and the second value denotes the angle to the vertical side. The default values are 45 degrees in radians. However, when **orthographic** field is set to TRUE, the **fieldOfView** field denotes the width and height of the near plane and far plane.

The **nearPlane** and **farPlane** fields specify the distances from the viewpoint to the near plane and far plane of the visibility area. The texture and depth data shows the area closed by the near plane, far plane and the **fieldOfView**. The depth data are scaled to the distance from **nearPlane** to **farPlane**.

The **splatMinMax** field specifies the minimum and maximum splat vector lengths. The splatU and splatV data of SimpleTextureV2 is scaled to the interval defined by the splatMinMax field.

The **orthographic** field specifies the view type of the IBR texture. When set to TRUE, the IBR texture is based on orthographic view. Otherwise, the IBR texture is based on perspective view.

The **position, orientation, fieldOfView, nearPlane, farPlane,** and **orthographic** fields are exposedField types, which are for extrinsic parameters. The DepthImage node supports the camera movement and changeable view frustum corresponding to movement or deformation of a DIBR object.

Reference images that are suitable to the characteristic of a DIBR model are obtained in the modeling stage. Therefore, the fields that reflect the camera movement and the changeable view frustum and the reference images in the modeling stage are used to create a view frustum and a DIBR object in the rendering stage.

4.5.3.3 SimpleTextureV2 node

4.5.3.3.1 Node interface

```
SimpleTextureV2 { #%NDT=SFDepthTextureNode
      field       SFTextureNode       texture          NULL
      field       SFTextureNode       depth            NULL
      field       SFTextureNode       normal           NULL
      field       SFTextureNode       splatU           NULL
      field       SFTextureNode       splatV           NULL
}
```

4.5.3.3.2 Functionality and semantics

The **SimpleTextureV2** node defines a single layer of IBR texture.

The **texture** field specifies the flat image that contains color for each pixel. It shall be one of the various types of texture nodes (**ImageTexture**, **MovieTexture** or **PixelTexture**).

The **depth** field specifies the depth for each pixel in the **texture** field. The size of the depth map shall be the same size as the image or movie in the **texture** field. Depth field shall be one of the various types of texture

© ISO/IEC 2006 – All rights reserved

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

nodes (ImageTexture, MovieTexture or PixelTexture), where only the nodes representing gray scale images are allowed. If the depth field is unspecified, the alpha channel in the texture field shall be used as the depth map. If the depth map is not specified through depth field or alpha channel, the result is undefined.

Depth field allows to compute the actual distance of the 3D points of the model to the plane which passes through the viewpoint and parallel to the near plane and far plane:

$$dist = nearPlane + \left(1 - \frac{d-1}{d_{\max}-1}\right)(farPlane - nearPlane).$$

Equation 13

where d is depth value and d_{max} is maximum allowed depth value. It is assumed that for the points of the model, $d>0$, where $d=1$ corresponds to far plane, $d=d_{max}$ corresponds to near plane.

This formula is valid for both perspective and orthographic case, since d is distance between the point and the plane. max d is the largest d value that can be represented by the bits used for each pixel:

(1) If the depth is specified through **depth** field, then depth value d equals to the gray scale.

(2) If the depth is specified through alpha channel in the image defined via **texture** field, then the depth value d is equal to alpha channel value.

The depth value is also used to indicate which points belong to the model: only the point for which d is nonzero belong to the model.

For animated DepthImage-based model, only DepthImage with SimpleTextures as diTextures are used.

Each of the Simple Textures can be animated in one of the following ways:

(1) **depth** field is still image satisfying the above condition, **texture** field is arbitrary MovieTexture

(2) **depth** field is arbitrary MovieTexture satisfying the above condition on the **depth** field, **texture** field is still image

(3) both **depth** and **texture** are MovieTextures, and **depth** field satisfies the above condition

(4) **depth** field is not used, and the depth information is retrieved from the alpha channel of the MovieTexture that animates the **texture** field

The **normal** field specifies the normal vector for each pixel in the **texture** field. The normal vector should be assigned to the object-space point sample derived from extruding the pixel with depth to 3-space. The normal map shall be the same size as the image or movie in the **texture** field. Normal field shall be one of the various types of texture nodes (ImageTexture, MovieTexture, or PixelTexture), where only the nodes representing color images are allowed. If the normal map is not specified through the normal field, the decoder can calculate a normal field by evaluating the cross-product of the splatU and splatV fields. If neither the normal map nor the splatU and splatV fields are specified, only basic rendering is possible.

The **splatU** and **splatV** fields specify the tangent plane and reconstruction kernel needed for high-quality point-based rendering. Both splatU and splatV fields have to be scaled to the interval defined by the splatMinMax field.

The **splatU** field specifies the splatU vector for each pixel in the **texture** field. The splatU vector should be assigned to the object-space point sample derived from extruding the pixel with depth to 3-space. The splatU map shall be the same size as the image or movie in the **texture** field. splatU field shall be one of the various types of texture nodes (ImageTexture, MovieTexture, or PixelTexture), where the nodes either representing color or gray scale images are allowed. If the splatU map is specified as gray scale image the decoder can calculate a circular splat by using the normal map to produce a tangent plane and the splatU map as radius. In this case, if the normal map is not specified, the result is undefined. If the splatU map is specified as color image, it can be used in conjunction with the splatV map to calculate a tangent frame and reconstruction

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

kernel for high-quality point-based rendering. If neither the normal map nor the splatV map is specified, the result is undefined.

The **splatV** field specifies the splatV vector for each pixel in the **texture** field. The splatV vector should be assigned to the object-space point sample derived from extruding the pixel with depth to 3-space. The splatV map shall be the same size as the image or movie in the **texture** field. splatV field shall be one of the various types of texture nodes (ImageTexture, MovieTexture, or PixelTexture), where only the nodes representing color images are allowed. If the splatU map is not specified as well, the result is undefined.

4.5.3.4 PointTextureV2 node

4.5.3.4.1 Node interface

PointTextureV2 { #%NDT=SFDepthTextureNode			
field	**SFInt32**	**width**	**256**
field	**SFInt32**	**height**	**256**
field	**SFInt32**	**depthNbBits**	**7**
field	**MFInt32**	**depth**	**[]**
field	**MFColor**	**color**	**[]**
field	**SFNormalNode**	**normal**	
field	**MFVec3f**	**splatU**	**[]**
field	**MFVec3f**	**splatV**	**[]**
}			

4.5.3.4.2 Functionality and semantics

The **PointTextureV2** node defines multiple layers of IBR points.

The **width** and **height** field specify the width and height of the texture.

Geometrical meaning of the depth values, and all the conventions on their interpretation adopted for the SimpleTexture, apply here as well.

The **depth** field specifies a multiple depths of each point in the projection plane, which is assumed to be farPlane (see above) in the order of traversal, which starts from the point in the lower left corner and traverses to the right to finish the horizontal line before moving to the upper line. For each point, the number of depths (pixels) is first stored and that number of depth values shall follow.

The **color** field specifies color of current pixel. The order shall be the same as the **depth** field except that number of depths (pixels) for each point is not included.

The **depthNbBits** field specifies the number of bits used for the original depth data. The value of depthNbBits ranges from 0 to 31, and the actual number of bits used in the original data is depthNbBits+1. The d_{max} used in the distance equation is derived as follows:

$$d_{\max} = 2^{(depthNbBits+1)} - 1.$$

Equation 14

The **normal** field specifies normals for each specified depth of each point in the projection plane in the same order. The normal vector should be assigned to the object-space point sample derived from extruding the pixel with depth to 3-space. If the normals are not specified through the normal field, the decoder can calculate a normal field by evaluating the cross-product of the splatU and splatV fields. If neither the normals nor the splatU and splatV fields are specified, only basic point rendering is possible. Normals can be quantized by using the SFNormalNode functionality.

The **splatU** field specifies splatU vectors for each specified depth of each point in the projection plane in the same order. The splatU vector should be assigned to the object-space point sample derived from extruding the pixel with depth to 3-space. If the splatV vectors are not specified the decoder can calculate a circular

© ISO/IEC 2006 – All rights reserved

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

splat by using the normals to produce a tangent plane and the length of the splatU vectors as radius. In this case, if the normals are not specified, the result is undefined. If the splatU vectors are specified, it can be used in conjunction with the splatV vectors to calculate a tangent frame and reconstruction kernel for high-quality point-based rendering. If neither the normals nor the splatV vectors are specified, the result is undefined.

The **splatV** field specifies splatV vectors for each specified depth of each point in the projection plane in the same order. The splatV vector should be assigned to the object-space point sample derived from extruding the pixel with depth to 3-space. If the splatU vectors are not specified as well, the result is undefined.

4.5.4 Multitexturing

4.5.4.1 MultiTexture Node

4.5.4.1.1 Node Interface

```
MultiTexture { #%NDT=SFTextureNode
  exposedField SFFloat          alpha        1       #[0,1]
  exposedField SFColor          color        1 1 1   #[0,1]
  exposedField MFInt            function     []
  exposedField MFInt            mode         []
  exposedField MFInt            source       []
  exposedField MFTextureNode    texture      []
  exposedField MFVec3f          cameraVector []
  exposedField SFBool           transparent  FALSE
}
```

4.5.4.1.2 Functionality and semantics

MultiTexture enables the application of several individual textures to a 3D object to achieve a more complex visual effect. MultiTexture can be used as a value for the texture field in an Appearance node.

The **texture** field contains a list of texture nodes (e.g., ImageTexture, PixelTexture, MovieTexture). The texture field may not contain another MultiTexture node.

The **cameraVector** field contains a list of camera vectors in 3D for each texture in the **texture** field. These vectors point from each associated camera to the 3D scene center. The view vectors are used to calculate texture weights according to the unstructured lumigraph approach from [85] for each render cycle, to weight all textures according to the actual scene viewpoint.

The **color** and **alpha** fields define base RGB and alpha values for SELECT mode operations.

The **mode** field controls the type of blending operation. The available modes include MODULATE for a lit Appearance, REPLACE for an unlit Appearance and several variations of the two. However, for view-dependent Multitexturing the default mode MODULATE shall be used in conjuction with the **source** field value “FACTOR”. Table 5 lists possible multitexture modes.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 5 — Multitexture modes.

VALUE	MODE	Description
00000	MODULATE	Multiply texture color with current color Arg1 × Arg2
00001	REPLACE	Replace current color Arg2
00010	MODULATE2X	Multiply the components of the arguments, and shift the products to the left 1 bit (effectively multiplying them by 2) for brightening.
00011	MODULATE4X	Multiply the components of the arguments, and shift the products to the left 2 bits (effectively multiplying them by 4) for brightening.
00100	ADD	Add the components of the arguments Arg1 + Arg2
00101	ADDSIGNED	Add the components of the arguments with a -0.5 bias, making the effective range of values from −0.5 through 0.5.
00110	ADDSIGNED2X	Add the components of the arguments with a -0.5 bias, and shift the products to the left 1 bit.
00111	SUBTRACT	Subtract the components of the second argument from those of the first argument. Arg1 − Arg2
01000	ADDSMOOTH	Add the first and second arguments, then subtract their product from the sum. Arg1 + Arg2 − Arg1 × Arg2 = Arg1 + (1 − Arg1) × Arg2
01001	BLENDDIFFUSEALPHA	Linearly blend this texture stage, using the interpolated alpha from each vertex. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
01010	BLENDTEXTUREALPHA	Linearly blend this texture stage, using the alpha from this stage's texture. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
01011	BLENDFACTORALPHA	Linearly blend this texture stage, using the alpha factor from the MultiTexture node. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
01100	BLENDCURRENTALPHA	Linearly blend this texture stage, using the alpha taken from the previous texture stage. Arg1 × (Alpha) + Arg2 × (1 − Alpha)
01101	MODULATEALPHA_ADDCOLOR	Modulate the color of the second argument, using the alpha of the first argument; then add the result to argument one. Arg1.RGB + Arg1.A × Arg2.RGB
01110	MODULATEINVALPHA_ADDCOLOR	Similar to MODULATEALPHA_ADDCOLOR, but use the inverse of the alpha of the first argument. (1 − Arg1.A) × Arg2.RGB + Arg1.RGB
01111	MODULATEINVCOLOR_ADDALPHA	Similar to MODULATECOLOR_ADDALPHA, but use the inverse of the color of the first argument. (1 − Arg1.RGB) × Arg2.RGB + Arg1.A
10000	OFF	Turn off the texture unit
10001	SELECTARG1	Use color argument 1 Arg1
10010	SELECTARG2	Use color argument 1 Arg2

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

© ISO/IEC 2006 – All rights reserved

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

VALUE	MODE	Description
10011	DOTPRODUCT3	Modulate the components of each argument (as signed components), add their products, then replicate the sum to all color channels, including alpha. This can do either diffuse or specular bump mapping with correct input. Performs the function (Arg1.R × Arg2.R + Arg1.G × Arg2.G + Arg1.B × Arg2.B) where each component has been scaled and offset to make it signed. The result is replicated into all four (including alpha) channels.
10100 – 11111		Reserved for future use

The **source** field determines the color source for the second argument. Table 6 lists valid values for the source field. For view-dependent Multitexturing “FACTOR” shall be used in conjuction with the **mode** field value MODULATE.

Table 6 — Values for the *source* field

VALUE	MODE	Description
000	"" (default)	The second argument color (ARG2) is the color from the previous rendering stage (DIFFUSE for first stage).
001	"DIFFUSE"	The texture argument is the diffuse color interpolated from vertex components during Gouraud shading.
010	"SPECULAR"	The texture argument is the specular color interpolated from vertex components during Gouraud shading.
011	"FACTOR"	The texture argument is the factor (color, alpha) from the MultiTexture node.
100-111		Reserved for future use

The **function** field defines an optional function to be applied to the argument after the mode has been evaluated. Table 7 lists valid values for the **function** field.

Table 7 — Values for the *function* field.

VALUE	MODE	Description
000	"" (default)	No function is applied.
001	"COMPLEMENT"	Invert the argument so that, if the result of the argument were referred to by the variable x, the value would be 1.0 minus x.
010	"ALPHAREPLICATE"	Replicate the alpha information to all color channels before the operation completes.
011-111		Reserved for future use

Mode may contain an additional Blending mode for the alpha channel; e.g., "MODULATE,REPLACE" specifies Color = (Arg1.color × Arg2.color, Arg1.alpha).

The number of used texture stages is determined by the length of the texture field. If there are fewer mode values, the default mode is "MODULATE".

Note: Due to the texture stage architecture and its processing order of textures in common graphic cards, the result of general texture weighting depends on the order of textures if more than two textures are used. If order-independent texture mapping is required, the proposed settings can be used, i.e. MODULATE for the **mode** field and “TFACTOR” for the **source** field.

Beside the **MultiTexture**-Node that assigns the actual 2D images to the scene, contains blending modes and transform parameters, the second component of Multi-Texturing is the **MultiTextureCoordinate**-Node. This node addresses the relative 2D coordinates of each texture, which are combined with the 3D points of the

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

scene geometry. In Multi-Texturing, one 3D point is associated with *n* 2D texture points with *n* being the number of views. The node syntax for **MultiTextureCoordinate** in X3D is as follows and can be used as is.

4.5.4.2 MultiTextureCoordinate Node

MultiTextureCoordinate node supplies multiple texture coordinates per vertex. This node can be used to set the texture coordinates for the different texture channels.

4.5.4.2.1 Node interface

```
MultiTextureCoordinate { #%NDT=SFTextureCoordinateNode
  exposedField MFTextureCoordinateNode texCoord []
}
```

4.5.4.2.2 Functionality and semantics

Each entry in the **texCoord** field may contain a **TextureCoordinate** or **TextureCoordinateGenerator** node.

By default, if using **MultiTexture** with an **IndexedFaceSet** without a **MultiTextureCoordinate texCoord** node, texture coordinates for channel 0 are replicated along the other channels. Likewise, if there are too few entries in the **texCoord** field, the last entry is replicated.

Example:

```
Shape {
  appearance Appearance {
    texture MultiTexture {
      mode [ 0 0 0 0 ]
      source [ 3 3 3 3 ]
      texture [
        ImageTexture { url "np00.jpg" }
        ImageTexture { url "np01.jpg" }
        ImageTexture { url "np02.jpg" }
        ImageTexture { url "np03.jpg" }
      ]
    }
  }
  geometry IndexedFaceSet {
      ...
    texCoord MultiTextureCoord {
      texCoord [
        TextureCoordinate { ... }
        TextureCoordinate { ... }
        TextureCoordinate { ... }
        TextureCoordinate { ... }
      ]
    }
  }
}
```

4.6 Modeling tools

Modeling tools consit of **NonLinearDeformer** and **FFD** nodes.

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

© ISO/IEC 2006 – All rights reserved

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

4.6.1 NonLinearDeformer

4.6.1.1 Node interface

```
NonLinearDeformer { #%NDT=SFGeometryNode
    exposedField  SFInt32     type     0
    exposedField  SFVec3f     axis     0 0 1
    exposedField  SFFloat     param    0
    exposedField  MFFloat     extend   []
    exposedField  SFGeometryNode  geometry   NULL
}
```

4.6.1.2 Functionality and semantics

type is the desired deformation (0: tapering, 1:twisting, 2:bending). **axis** is the axis along which the deformation is performed, **param** the parameter of the transformation, **extend** its bounds, and **geometry** the geometry node on which is the deformation is performed or another **NonLinearDeformer** node so to chain the transformations.

Table 8 — Semantic of param and extend values for each deformation type.

	Type	*param*	*Extend*
0	tapering	Radius	[relative position, relative radius]*
1	twisting	Angle	[Angle min, angle max]
2	bending	Curvature	[Curvature min, curvature max, y min, y max]

For tapering, **extend** consists of a serie of 2 values: the first is the position at which the radius should be. This way a profile can be defined. The relative position along the axis of the transformation in object space: 0% at the beginning, and 100% at the end. The radius is relative to the **param** and is given in percentage. **extend** is used similarly for the other transformations.

Transformations are given by [13]:

Tapering

To taper an object long the z-axis, x- and y-axes are just scales as a function of z:

$$(X, Y, Z) = (rx, ry, z) \quad \text{and} \quad r = f(z)$$

where $f(z)$ specifies the rate of scale per unit length along the z-axis and can be a linear or nonlinear tapering profile or function.

Twisting

To rotate an object through an angle θ about the z-axis:

$$(X, Y, Z) = (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta, z) \quad \text{and} \quad \theta = f(z)$$

where $f(z)$ specifies the rate of twist per unit length along the z-axis.

Bending

A global linear bend along an axis is a composite transformation comprising a bent region and a region outside the bent region where the deformation is a rotation and a translation. Barr defines a bend region along

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the y-axis as: $y_{\min} \le y \le y_{\max}$. The radius of curvature of the bend is k^{-1} and the center of the bend is at $y = y_0$. The bending angle is: $\theta = k(y' - y_0)$, where

$$y' = \begin{cases} y_{\min} & \text{if} \quad y \le y_{\min} \\ y & \text{if} \quad y_{\min} \le y < y_{\max} \\ y_{\max} & \text{if} \quad y \ge y_{\max} \end{cases}$$

The deformation is given by

$$\begin{aligned} X &= x \\ Y &= \begin{cases} -\sin\theta(z - k^{-1}) + y_0 & y_{\min} \le y \le y_{\max} \\ -\sin\theta(z - k^{-1}) + y_0 + \cos\theta(y - y_{\min}) & y < y_{\min} \\ -\sin\theta(z - k^{-1}) + y_0 + \cos\theta(y - y_{\max}) & y > y_{\max} \end{cases} \\ Z &= \begin{cases} \cos\theta(z - k^{-1}) + k^{-1} & y_{\min} \le y \le y_{\max} \\ \cos\theta(z - k^{-1}) + k^{-1} + \sin\theta(y - y_{\min}) & y < y_{\min} \\ \cos\theta(z - k^{-1}) + k^{-1} + \sin\theta(y - y_{\max}) & y > y_{\max} \end{cases} \end{aligned}$$

4.6.2 Free-form deformations

4.6.2.1 Node interface

```
FFD { #%NDT=SF3DNode
    eventIn         MF3DNode addChildren
    eventIn         MF3DNode removeChildren
    exposedField    MF3DNode children    []

    field           SFInt32  uDimension  2   # [2, 257]
    field           SFInt32  vDimension  2   # [2, 257]
    field           SFInt32  wDimension  2   # [2, 257]
    field           MFFloat  uKnot       []  # (-∞,∞)
    field           MFFloat  vKnot       []  # (-∞,∞)
    field           MFFloat  wKnot       []  # (-∞,∞)
    field           SFInt32  uOrder      2   # [2, 33]
    field           SFInt32  vOrder      2   # [2, 33]
    field           SFInt32  wOrder      2   # [2, 33]
    exposedField    MFVec4f  controlPoint []

}
```

4.6.2.2 Functionality and semantics

The node definition is the same as for **NURBSSurface** in subclause 4.3.2 (except for the bounds and default values of the Dimenison and Order fields) and the first 3 fields are as for a **Group** node: they enable to define the scene embedded in the FFD space.

A FFD node acts only on a scene on the same level in the transform hierarchy because a FFD applies only on vertices of shapes. If an object is made of many shapes, there are nested **Transform** nodes. If we pass solely the DEF of these nodes, then we have no notion of what the transforms applied to the nodes are. By passing the DEF of a grouping node, which encapsulates the scene to be deformed, we can effectively calculate the transformation applied on a node.

© ISO/IEC 2006 – All rights reserved

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

EXAMPLE

```
# The control points of a FFD are animated. The FFD encloses two shapes, which are
deformed
# as the control points move.
DEF TS TimeSensor {}
DEF PI CoordinateInterpolator4D {
   key [ … ]
   keyValue [ … ]
}

DEF BoxGroup Group {
   children [ Shape { geometry Box {} } ]
}

DEF SkeletonGroup Group {
   children [
      …# describe here a full skeleton
   ]
}
DEF FFDNode FFD {
   …# specify NURBS deformation volume
   children [
      USE BoxGroup
      USE SkeletonGroup
   ]
}

ROUTE TS.fraction_changed TO PI.set_fraction
ROUTE PI.value_changed TO FFDNode.controlPoint
```

4.7 Generic skeleton, muscle and skin-based model definition and animation

This subclause defines a generic animation framework for models based on skeletons and muscles. In order to represent the model (geometry, appearance and semantic information) in the scene the following nodes are defined: **SBBone**, **SBSegment**, **SBSite**, **SBMuscle**, **SBSkinnedModel and SBVCAnimation**. The animation parameters are descripted in subclause 4.7.2

4.7.1 Generic skeleton, muscle and skin-based model definition

4.7.1.1 SBBone

4.7.1.1.1 Node interface

SBBone{ #%NDT=SFSBBoneNode, SF3DNode, SF2DNode

eventIn	**MF3DNode**	**addChildren**	
eventIn	**MF3DNode**	**removeChildren**	
exposedField	**SFInt32**	**boneID**	**0**
exposedField	**MFInt32**	**skinCoordIndex**	**[ ]**
exposedField	**MFFloat**	**skinCoordWeight**	**[ ]**
exposedField	**SFVec3f**	**endpoint**	**0 0 1**
exposedField	**SFInt32**	**falloff**	**1**
exposedField	**MFFloat**	**sectionPosition**	**[ ]**
exposedField	**MFFloat**	**sectionInner**	**[ ]**
exposedField	**MFFloat**	**sectionOuter**	**[ ]**
exposedField	**SFInt32**	**rotationOrder**	**0**
exposedField	**MFNode**	**children**	**[ ]**
exposedField	**SFVec3f**	**center**	**0 0 0**
exposedField	**SFRotation**	**rotation**	**0 0 1 0**
exposedField	**SFVec3f**	**translation**	**0 0 0**
exposedField	**SFVec3f**	**scale**	**1 1 1**

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    exposedField  SFRotation     scaleOrientation   0 0 1 0
    exposedField  SFInt32        ikChainPosition    0
    exposedField  MFFloat        ikYawLimit         [ ]
    exposedField  MFFloat        ikPitchLimit       [ ]
    exposedField  MFFloat        ikrollLimit        [ ]
    exposedField  MFFloat        ikTxLimit          [ ]
    exposedField  MFFloat        ikTyLimit          [ ]
    exposedField  MFFloat        ikTzLimit          [ ]
}
```

4.7.1.1.2 Functionality and semantics

SBBone node specifies data related to one bone from the skeleton.

The **boneID** field is a unique identifier which allows that the bone to be addressed at animation run-time.

The **center** field specifies a translation offset from the origin of the local coordinate system.

The **translation** field specifies a translation to the bone coordinate system.

The **rotation** field specifies a rotation of the bone coordinate system.

The **scale** field specifies a non-uniform scale of the bone coordinate system. **scale** values shall be greater than zero.

The **scaleOrientation** specifies a rotation of the bone coordinate system before the scale (to specify scales in arbitrary orientations). The **scaleOrientation** applies only to the scale operation.

The possible geometric 3D transformation consists of (in order): 1) (possibly) non-uniform scale about an arbitrary point, 2) a rotation about an arbitrary point and axis and 3) a translation.

The **rotationOrder** field specifies the rotation order when deals with the decomposition of the rotation in respect with system coordinate axes.

Two ways of specifying the influence region of the bone are allowed:

f) The **skinCoordIndex** field contains a list of indices of all skin vertices affected by the current bone. Mostly, the skin influence region of bone will contain vertices from the 3D neighborhood of the bone, but special cases of influence are also accepted.

1) The **skinCoordWeight** field contains a list of weights (one per vertex listed in **skinCoordIndex**) that measures the contribution of the current bone to the vertex in question. The length **skinCoordIndex** is equal with the length of **skinCoordWeight**. The sum of all **skinCoordWeight** related to the same vertex must be 1.

2) The **endpoint** field specifies the bone 3D end point and is used to compute the bone length.

3) The **sectionInner** field is a list of inner influence region radii for different sections.

4) The **sectionOuter** field is a list of outer influence region radii for different sections.

5) The **sectionPosition** field is a list of positions of all the sections defined by the designer.

6) The **falloff** field specifies the function between the amplitude affectedness and distance : -1 for x^3, 0 for x^2, 1 for x, 2 for $\sin(\frac{\pi}{2}x)$, 3 for $\sqrt{x}$ and 4 for $\sqrt[3]{x}$.

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

© ISO/IEC 2006 – All rights reserved

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

The two schemes can be used independently or in combination, in which case the individual vertex weights take precedence.

The **ikChainPosition** field specifies the position of the bone in the kinematics chain. If the bone is the root of the IK chain then **ikChainPosition**=1. In this case, when applying IK scheme, only the orientation of the bone is changed. If the bone is last in the kinematics chain **ikChainPosition**=2. In this case, the animation stream has to include the desired position of the bone (X, Y and Z world coordinates). If **ikChainPosition**=3 the bone belongs to the IK chain but is not the first or the last one in the chain. In this case, position and orientation of the bone are computed by the IK procedure. Finally, if the bone does not belong to any IK chain (**ikChainPosition**=0), it is necessary to transmit the bone local transformation in order to animate the bone. If an animation stream contains motion information about a bone which has **ikChainPosition** 1, this information will be ignored. If an animation stream contains motion information about a bone which has **ikChainPosition** 3, this means that the animation producer wants to ensure the orientation of the bone and the IK solver will use this value as a constrain.

The **ikYawLimit** field consists in a pair of min/max values which limit the bone rotation with respect to the X axis.

The **ikPitchLimit** field consists in a pair of min/max values which limit the bone rotation with respect to the Y axis.

The **ikRollLimit** field consists in a pair of min/max values which limit the bone rotation with respect to the Z axis.

The **ikTxLimit** field consists in a pair of min/max values which limit the bone translation in the X direction.

The **ikTyLimit** field consists in a pair of min/max values which limit the bone translation in the Y direction.

The **ikTzLimit** field consists in a pair of min/max values which limit the bone translation in the Z direction.

The **SBBone** node is used as a building block to describe the hierarchy of the articulated model by attaching one or more child objects. The **children** field has the same semantic as used in ISO/IEC 14496-11; the absolute geometric transformation of any child of a bone is obtained through a composition with the bone-parent transformation.

4.7.1.2 SBSegment

4.7.1.2.1 Node interface

```
SBSegment{ #%NDT=SFSBSegmentNode, SF3DNode, SF2DNode
    eventIn       MFNode   addChildren
    eventIn       MFNode   removeChildren
    exposedField  SFString name              ""
    exposedField  SFVec3f  centerOfMass      0 0 0
    exposedField  MFVec3f  momentsOfInertia  [ 0 0 0 0 0 0 0 0 0 ]
    exposedField  SFFloat  mass              0
    exposedField  MFNode   children          [ ]
}
```

4.7.1.2.2 Functionality and semantics

The **name** field must be present, so that the **SBSegment** can be identified at runtime. Each **SBSegment** should have a DEF name that matches the **name** field for that Segment, but with a distinguishing prefix in front of it.

The **mass** field is the total mass of the segment.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **centerOfMass** field is the location within the segment of its center of mass. Note that a zero value was chosen for the mass in order to give a clear indication that no mass value is available.

The **momentsOfInertia** field contains the moment of inertia matrix. The first three elements are the first row of the 3x3 matrix, the next three elements are the second row, and the final three elements are the third row.

The **children** field can be any object attached at this level of the skeleton, including a **SBSkinnedModel**.

An **SBSegment** node is a grouping node especially introduced to address two issues:

g) The first one is to the requirement to separate different parts from the skinned model into deformation-independent parts. Between two deformation-independent parts the geometrical transformation of one of them do not imply skin deformations on the other. This is essential for run-time animation optimization. The **SBSegment** node may contain as a child an **SBSkinnedModel** node (see the **SBSkinnedModel** node description below). Portions of the model which are not part of the seamless mesh can be attached to the skeleton hierarchy by using an **SBSegment** node;

h) The second deals with the requirement to attach standalone 3D objects at different parts of the skeleton hierarchy. For example, a ring can be attached to a finger; the ring geometry and attributes are defined outside of skinned model but the ring will have the same local geometrical transformation as the attached bone.

4.7.1.3 SBSite

4.7.1.3.1 Node interface

```
SBSite {#%NDT=SFSBSiteNode, SF3DNode, SF2DNode
    eventIn       MF3DNode    addChildren
    eventIn       MF3DNode    removeChildren
    exposedField  SFVec3f     center           0 0 0
    exposedField  MF3DNode    children         [ ]
    exposedField  SFString    name             ""
    exposedField  SFRotation  rotation         0 0 1 0
    exposedField  SFVec3f     scale            1 1 1
    exposedField  SFRotation  scaleOrientation 0 0 1 0
    exposedField  SFVec3f     translation      0 0 0
}
```

4.7.1.3.2 Functionality and semantics

The **center** field specifies a translation offset and can be used to compute a bone length. The **rotation** field specifies a rotation of the coordinate system of the **SBSite** node.

The **scale** field specifies a non-uniform scale of the **SBSite** node coordinate system and the **scale** values must be greater than zero.

The **scaleOrientation** specifies a rotation of the coordinate system of the **SBSite** node before the scale thus allowing a scale at an arbitrary orientation. The **scaleOrientation** applies only to the scale operation.

The **translation** field specifies a translation of the coordinate system of the **SBSite** node.

The **children** field is used to store any object that can be attached to the **SBSegment** node.

The **SBSite** node can be used for three purposes. The first is to define an "end effector", i.e. a location which can be used by an inverse kinematics system. The second is to define an attachment point for accessories such as clothing. The third is to define a location for a virtual camera in the reference frame of a **SBSegment** node.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SBSite nodes are stored within the **children** field of an **SBSegment** node. The **SBSite** node is a specialized grouping node that defines a coordinate system for nodes in its **children** field that is relative to the coordinate systems of its parent node. The reason a **SBSite** node is considered a specialized grouping node is that it can only be defined as a child of a **SBSegment** node.

4.7.1.4 SBMuscle

4.7.1.4.1 Node interface

```
SBMuscle{ #%NDT=SFSBMuscleNode, SF3DNode, SF2DNode
    exposedField  MFInt32    skinCoordIndex    [ ]
    exposedField  MFFloat    skinCoordWeight   [ ]
    exposedField  SFNode     muscleCurve       NULL
    exposedField  SFInt32    muscleID          0
    exposedField  SFInt32    radius            1
    exposedField  SFInt32    falloff           1
}
```

4.7.1.4.2 Functionality and semantics

The **skinCoordIndex** field consists of a list of vertex indices from the skinned model skin which are affected by the “muscle”

The **skinCoordWeight** field consists of a list of weights indicating in what measure a vertex in affected by the “muscle”

The **muscleCurve** field is a NurbCurve as defined in subclause 4.3.2.2.

The **radius** field specifies the maximum distance where the “muscle” will affect the skin

The **falloff** field specifies the function between the amplitude affectedness and distance : -1 for x^3, 0 for x^2, 1 for x, 2 for $\sin(\frac{\pi}{2}x)$, 3 for $\sqrt{x}$ and 4 for $\sqrt[3]{x}$.

Performing deformation consists in affecting the form of the muscleCurve by 1) affecting the position of the control points of the curve, 2) affecting the weight of control points or/and 3) affecting the *knot* sequence.. Depending on the author, the animation stream can contain one animation mechanism or a combination of 1), 2) and 3).

At the modeling stage, each affected vertex v_i from the skin is assigned a point $v_i{}^c$ from the curve, as the closest point. During animation, the translation of $v_i{}^c$ obtained from the update values of **controlPoint**, **weight** or/and **knot** fields, will induce a translation on v_i :

— **skinCoordWeight** field is specified for vertex v_i, then:

$$Tv_i = skinCoordWeight[k] * Tv_i{}^c,$$

where *k* is the index of vertex v_i in the model vertices index list;

— **radius** field is specified, then $Tv_i = f(\frac{radius - d(v_i, v_i^c)}{radius}) * Tv_i^c$

with $f(\ )$ specified by the falloff field.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.7.1.5 SBSkinnedModel

4.7.1.5.1 Node interface

```
SBSkinnedModel{ #%NDT=SF3DNode,SF2DNode
    exposedField  SFString        name                          ""
    exposedField  SFVec3f         center                        0 0 0
    exposedField  SFRotation      rotation                      0 0 1 0
    exposedField  SFVec3f         translation                   0 0 0
    exposedField  SFVec3f         scale                         1 1 1
    exposedField  SFRotation      scaleOrientation              0 0 1 0
    exposedField  MF3DNode            skin                          [ ]
    exposedField  SFCoordinateNode    skinCoord                     NULL
    exposedField  SFNormalNode        skinNormal                    NULL
    exposedField  MF3DNode            skeleton                      [ ]
    exposedField  MFBoneNode          bones                         [ ]
    exposedField  MF3DNode            muscles                       [ ]
    exposedField  MFSegmentNode       segments                      [ ]
    exposedField  MFSiteNode          sites                         [ ]
    exposedField  SF3DNode            weighsComputationSkinCoord    NULL
}
```

4.7.1.5.2 Functionality and semantics

The **SBSkinnedModel** node is the top of the hierarchy of Skin&Bones related nodes and contains the definition parameters for the entire seamless model or of a seamless part of the model.

The **name** field specify the name of the skinned model allowing easily identification at the animation run-time.

The **center** field specifies a translation offset from the origin of the local coordinate system.

The **translation** field specifies a translation of the coordinate system.

The **rotation** field specifies a rotation of the coordinate system.

The **scale** field specifies a non-uniform scale of the coordinate system. **scale** values shall be greater than zero.

The **scaleOrientation** specifies a rotation of the coordinate system before the scale (to specify scales in arbitrary orientations). The **scaleOrientation** applies only to the scale operation.

The **skinCoord** contains the 3d coordinates of all vertices of the seamless model.

The **skin** consists of a collection of shapes that share the same **skinCoord**. This mechanism allows considering the model as a continuous mesh and, in the same time, to attach different attributes (like color, texture) to different parts of the model.

The **skeleton** field specifies the root of the bones hierarchy.

The **bones** fields consist in the lists of all bones previously defined as **SBBone** node.

The **segments** fields consist in the lists of all bones previously defined as **SBSegment** node.

The **sites** fields consist in the lists of all bones previously defined as **SBSites** node. The **muscles** fields consist in the lists of all bones previously defined as **SBMuscle** node.

The **weighsComputationSkinCoord** field describes a specific static position of the skinned model. In many cases the static position of the articulated model defined by **skinCoord** and **skin** fields is not appropriate to

© ISO/IEC 2006 – All rights reserved

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

compute the influence region of a bone. In this case the **weighsComputationSkinCoord** field allows specifying the skinned model vertices in a more appropriate static posture. This posture will be used just during the initialization stage and ignored during the animation. All the skeleton transformations are related to the posture defined by **skinCoord** field.

4.7.1.6 SBVCAnimation

This node allows to group together a set of skinned models; in order to animate them the animation data is extracted from the same resource (file or strem).

4.7.1.6.1 Node interface

```
SBVCAnimation{ #%NDT=SF3DNode,SF2DNode
    exposedField  MFNode      virtualCharacters      [ ]
    exposedField  MFURL       url                    [ ]
}
```

4.7.1.6.2 Functionality and semantics

The **SBVCAnimation** node is a grouping node, which allows to attach a list of virtual characters to an animation stream.

The **virtualCharacters** field specifies a list of SBSkinnedModel nodes. The length of the list can be 1 or greater.

The **url** field refers to the BBA stream which contains encoded animation data related to the SBSkinnedModel nodes from the VirtualCharacters list and is used for outband bitstreams. The animation will be extracted from the first element of the animationURL list and if the case when it is not available the following element will be used.

4.7.1.7 SBVCAnimationV2

4.7.1.7.1 Introduction

This node is an extension of the SBVCAnimation node and the added functionality consists in streaming control and animation data collection. The BBA stream can be controlled as a elementary media stream, and can be used in connection with the MediaControl node.

4.7.1.7.2 Syntax

```
SBVCAnimationV2{ #%NDT=SF3DNode,SF2DNode
        exposedField    MFNode    virtualCharacters    [ ]
        exposedField    MFURL     url                  [ ]
        exposedField    SFBool    loop                 FALSE
        exposedField    SFFloat   speed                1.0
        exposedField    SFTime    startTime            0
        exposedField    SFTime    stopTime             0
        eventOut        SFTime    duration_changed
        eventOut        SFBool    isActive
        exposedField    MFInt     activeUrlIndex       []
        exposedField    SFFloat   transitionTime       0
}
```

4.7.1.7.3 Semantics

The **virtualCharacters** field specifies a list of SBSkinnedModel nodes. The length of the list can be 1 or greater.

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The **url** field refers to the BBA stream which contains encoded animation data related to the SBSkinnedModel nodes from the virtualCharacters list and is used for outband bitstreams. The animation will be extracted from the first element of the animation URL list and if the case when it is not available the following element will be used.

The **loop**, **startTime**, and **stopTime** exposedFields and the **isActive** eventOut, and their effects on the SBVCAnimationV2 node, are similar with the ones described by VRML specifications (ISO/IEC 14772-1:1997) for AudioClip, MovieTexture, and TimeSensor nodes and are described as follows.

The values of the exposedFields are used to determine when the node becomes active or inactive.

The SBVCAnimationV2 node can execute for 0 or more cycles. A cycle is defined by field data within the node. If, at the end of a cycle, the value of **loop** is FALSE, execution is terminated. Conversely, if **loop** is TRUE at the end of a cycle, a time-dependent node continues execution into the next cycle. A time-dependent node with **loop** TRUE at the end of every cycle continues cycling forever if **startTime** >= **stopTime**, or until **stopTime** if **startTime** < **stopTime**.

The SBVCAnimationV2 node generates an **isActive** TRUE event when it becomes active and generates an **isActive** FALSE event when it becomes inactive. These are the only times at which an **isActive** event is generated. In particular, **isActive** events are not sent at each tick of a simulation.

The SBVCAnimationV2 node is inactive until its **startTime** is reached. When time *now* becomes greater than or equal to **startTime***,* an **isActive** TRUE event is generated and the SBVCAnimationV2 node becomes active (*now* refers to the time at which the player is simulating and displaying the virtual world). When a SBVCAnimationV2 node is read from a mp4 file and the ROUTEs specified within the mp4 file have been established, the node should determine if it is active and, if so, generate an **isActive** TRUE event and begin generating any other necessary events. However, if a SBVCAnimationV2 node would have become inactive at any time before the reading of the mp4 file, no events are generated upon the completion of the read.

An active SBVCAnimationV2 node will become inactive when **stopTime** is reached if **stopTime** > **startTime**. The value of **stopTime** is ignored if **stopTime** <= **startTime**. Also, an active SBVCAnimationV2 node will become inactive at the end of the current cycle if **loop** is FALSE. If an active SBVCAnimationV2 node receives a *set_loop* FALSE event, execution continues until the end of the current cycle or until **stopTime** (if **stopTime** > **startTime**), whichever occurs first. The termination at the end of cycle can be overridden by a subsequent *set_loop* TRUE event.

Any *set_startTime* events to an active SBVCAnimationV2 node are ignored. Any *set_stopTime* event where **stopTime** <= **startTime** sent to an active SBVCAnimationV2 node is also ignored. A *set_stopTime* event where **startTime** < **stopTime** <= *now* sent to an active SBVCAnimationV2 node results in events being generated as if **stopTime** has just been reached. That is, final events, including an **isActive** FALSE, are generated and the node becomes inactive. The **stopTime**_*changed* event will have the *set_stopTime* value.

A SBVCAnimationV2 node may be restarted while it is active by sending a *set_stopTime* event equal to the current time (which will cause the node to become inactive) and a *set_startTime* event, setting it to the current time or any time in the future. These events will have the same time stamp and should be processed as *set_stopTime,* then *set_startTime* to produce the correct behaviour.

The **speed** exposedField controls playback speed. It does not affect the delivery of the stream attached to the **SBVCAnimationV2** node. For streaming media, value of **speed** other than 1 shall be ignored.

A **SBVCAnimationV2** shall display first frame if **speed** is 0. For positive values of **speed**, the frame that an active **SBVCAnimationV2** will display at time *now* corresponds to the frame at animation time (i.e., in the animation's local time base with frame 0 at time 0, at speed = 1):

fmod (now - **startTime**, duration/**speed**)

If **speed** is negative, then the frame to display is the frame at animation time:

duration + fmod(now - **startTime**, duration/**speed**).

© ISO/IEC 2006 – All rights reserved

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

When a **SBVCAnimationV2** becomes inactive, the frame corresponding to the time at which the **SBVCAnimationV2** became inactive shall persist. The **speed** exposedField indicates how fast the movie shall be played. A speed of 2 indicates the animation plays twice as fast. Note that the **duration_changed** eventOut is not affected by the **speed** exposedField. **set_speed** events shall be ignored while the animation is playing.

An event shall be generated via the **duration_changed** field whenever a change is made to the **startTime** or **stopTime** fields. An event shall also be triggered if these fields are changed simultaneously, even if the duration does not actually change.

activeUrlIndex allows to select or to combine specific animation resource referred in the **url[]** field. When this field is instantiated the behavior of the **url[]** field changes from the alternative selection into a combined selection. In the case of alternative mode, if the first resource in the **url[]** field is not available, the second one will be used, and so on. In the combined mode the following cases can occur:n

(1) **activeUrlIndex** has one field: the resource from **url[]** that has this index is used for animation. When the **activeUrlIndex** is updated a transition between to the old animation (frame) and the new one is performed. The transition use linear interpolation for translation, center and scale and SLERP for spherical data as rotation and scaleOrientation. The time of transition is specified by using the **transitionTime** field.

(2) **activeUrlIndex** has several fields: a composition between the two resources is performed by the terminal: for the bones that are common in two or more resources a mean procedure has to be applied. The mean is computed by using linear interpolation for translation, center and scale and SLERP for spherical data as rotation and scaleOrientation.

In all the cases, when a transition between two animation resources is needed, when the transitionTime is not zero, the player must perform an interpolation. The transitionTime is specified in miliseconds.

4.7.2 Generic skeleton, muscle and skin-based model animation

Animating a 2D/3D articulated model requires knowledge of the position of each model vertex at each key-frame. Specifying such a data is an enormous and expensive task. For this reason the AFX animation system employs the bone-based modeling of articulated models which effectively attaches the model vertices to a bone hierarchy (skeleton).Moreover, a set of curve-based muscles allow to add local deformation of the skin. This technique avoids the necessity to specify the position for each vertex, only the local transformation of each bone in the skeleton and the muscle form being animated. The local bone transformation components (*translation*, *rotation*, *scale*, *scaleOrientation* and *center*) and the muscle-curve components (control points position and weights) are specified at each frame and, at the vertex level, the transformation is obtained by using the bone-vertex and muscle-vertex influence region.

To address streamed animation, the animation data is considered separately (independent of the model definition) and it will be specified for each key-frame.

Animating a skinned model is achieved through updates of the geometric transformation component of the skeleton by transforming the bones and/or to the muscle curve form.

A general transformation of a bone, as defined by the **SBBone** node, involves: translation in any direction, rotation with respect to any rotation axis, and scaling with respect to any direction. In the **SBBone** node definition the *rotation* field is defined as a SFRotation. In order to update the orientation of a bone the *rotation* field must by updated.

Within the animation resource the bone rotation is represented as a decomposition with respect with three axes. Bijectivity of the transformation between the angle-based notation and rotation matrix or quaternion representation is ensured by the *rotationOrder* field. A triplet of angles $[\theta_1, \theta_2, \theta_3]$ describes how a coordinate frame *r* rotates with respect to a static frame *s*, here, how a bone frame rotates with respect to its parent frame. The triplet is interpreted as a rotation by θ_1 around an axis A_1, then a rotation by θ_2 around an axis A_2, and finally a rotation by θ_3 around an axis A_3, with A_2 different from both A_1 and A_3. The axes are restricted to the

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

coordinate axes, X, Y, and Z, giving 12 possibilities: XYZ, XYX, YZX, YZY, ZXY, ZXZ, XZY, XZX, YXZ, YXY, ZYX, ZYZ. By considering the axis either in the bone frame (*r*) or its parent frame (*s*), there are 24 possible values for *rotationOrder*.

The bone-base animation (BBA) of a skinned model is performed by frame update of the **SBBone** transformation fields (**translation**, **rotation**, **scale**, **center** and **scaleOrientation**) and/or by updating the **SBMuscle** curve control points position, control points weight or (and) knot sequence. The BBA stream contains all the animation frames or just the data at the temporal key frames. In the last case the decoder will compute the intermediate frames by temporal interpolation. Linear interpolation is used for **translation** and **scale** components and linear quaternion interpolation is used for **rotation** and **scaleOrientation** components.

Each key-frame contains two fields: an animation mask vector and an animation values vector.

The animation mask vector consists in:

- a natural integer **NumberOfInterpolatedFrames** which indicates to the decoder the number of frames that have to be obtained by interpolation. If zero, the decoder interprets the received frame as a normal frame and sends it to the animation engine; if not the decoder computes **NumberOfInterpolatedFrames** intermediate frames and sends them to the animation engine, as well as the content of the received key-frame.

- the number of bones animated in the current frame; the number of muscles animated in the current frame.

- the **boneID** as well as the animation mask for each animated bone. See below for the description of **SBBone** animation mask.

- the **muscleID** as well as the animation mask for each animated muscle. See below for the description of **SBMuscle** animation mask.

The animation values vector consists in:

- the new values for each bone transformation component which need updating.

- the new values for each muscle control point which have been translated or change the weigh and the new values of the knot sequence of NURBS curve.

A BBA file or stream can contain the information related to a maximum number of 1024 **SBBone** and 1024 **SBMuscle** nodes belonging to one or more skinned models and grouped under the same SBVCAnimation node. The identifiers fields **boneID** and **muscleID** must be unique within an SBVCAnimation node and must be in the range [0 …1023].

Two representation of the animation data are supported by the standard: uncompressed format and compressed format.

Annex E describes the uncompressed animation file format. Subclause 5.4 shows the syntax of the compressed animation stream.

5 Bitstream specification

5.1 Wavelet Subdivision Surfaces

5.1.1 Downstream syntax

This is the syntax of the WaveletSubdivisionSurface downstream.

© ISO/IEC 2006 – All rights reserved

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

5.1.1.1 WMDecoderConfig

5.1.1.1.1 Syntax

```
class WMDecoderConfig extends AFXExtDescriptor : bit(8) tag = 2 {
   bit(1) hasScaleCoeff;
   if (hasScaleCoeff)
      int(5) NbBpSC;
   int(5) NbBPX;
   int(5) NbBPY;
   int(5) NbBPZ;
   int(2) Wtype;
   bit(1) lift;
   int(4) NbLevels;
   float(32) Xmax;
   float(32) Ymax;
   float(32) Zmax;
}
```

5.1.1.1.2 Semantics

hasScaleCoeff: thisis a flag with value 1 if and only if the scale coefficients are transmitted (otherwise they are supposed to be added to the base mesh vertices).

NbBpSC: this is the number of bits on which the scale coefficients will be read.

NbBPX, **NbBPY**, **NbBPZ**: these are respectively the numbers of bitplanes on which are encoded the magnitude of the first second and third component of the wavelet coefficients, an extra bit being necessary to encode their sign.

Wtype: this is an integer representing the wavelet type. 0 corresponds to midpoint subdivision, 1 to Loop's scheme, 2 to Dyn's butterfly scheme, and 3 is reserved for future use.

lift: this is a flag with value 1 if and only if the high pass filters are lifted [81].

NbLevels: this is the number of subdivision levels of the mesh.

Xmax, **Ymax**, **Zmax**: these are 32 bits floating point values representing the symmetric quantization intervals for the first, second and third coordinate respectively. These values are chosen so that, e.g., the first component of all wavelet coefficients belong to [-**Xmax**, +**Xmax**], and are therefore mapped to [$-2^{\mathbf{NbBPX}}+1$, $2^{\mathbf{NbBPX}}-1$].

5.1.1.2 Wavelet_Mesh_Object

5.1.1.2.1 Syntax

```
class Wavelet_Mesh_Object {
   bit(1) isInBand;
   if (isInBand)
      WMDecoderConfig WMDecoderConfig;
   bit(1) WMOL;
   bit(1) isInLocalCoordinates;
   if (WMOL && WMDecoderConfig.hasScaleCoeff)
      Wavelet_Mesh_Object_Scale_Coeff Coefficients;
   ReadZT ZeroTree;
}
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.1.1.2.2 Functionality and semantics

WMOL: a boolean with value 0 if and only if the current stream is a base layer.

isInLocalCoordinates: this is a flag with value 1 if and only if the decoder must reconstruct the mesh considering the wavelet coefficients in local frames.

ZeroTree: this is the SPIHT encoded representation of one or more bitplanes.

The decoding of the wavelet coefficients can start even if no base layer is received. It is then considered that the scale coefficients are all zero and the hierarchy is the default one.

5.1.1.3 Wavelet_Mesh_Object_Scale_Coeff

This is the class containing the scale coefficients.

5.1.1.3.1 Syntax

```
class Wavelet_Mesh_Object_Scale_Coeff {
   int i;
   for (i=0 ; i< Nb_sc ; i++) {
      int(WMDecoderConfig.NbBpSC) Sc[i];
   }
}
```

5.1.1.3.2 Semantics

Nb_sc: this is the number of scaling coefficients, known as the number of vertices in the base mesh.

Sc: this is an array containing the scale coefficients related to the vertices of the base mesh in the same order than in its description.

5.1.1.4 ReadZT

5.1.1.4.1 Syntax

```
class ReadZT {
   int FT, LT, i, j, k;
   int(5) LengthNbBits;
   int(5) FBP;
   int(5) LBP;
   bit(1) X;
   bit(1) Y;
   bit(1) Z;
   bit(1) isPartial;
   if (isPartial) {   // Partial transmission
      int(16) ZtId;
      FT = ZtId;
      LT = FT + 1
   } else {           // Full transmission
      FT = 0;
      LT = NumTree;
   }
   for (j=FBP; j<=LBP ; j++) {
      for (i=FT; i<LT; i++) {
         if (X) {
            int(LengthNbBits) BPLength;
            for (k=0 ; k<BPLength ; k++) {
               bit(1) Bx[i][j][k];
            }
         }
         if (Y) {
```

© ISO/IEC 2006 – All rights reserved

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

```
            int(LengthNbBits) BPLength;
            for (k=0 ; k<BPLength ; k++) {
               bit(1) By[i][j][k];
            }
         }
         if (Z) {
            int(LengthNbBits) BPLength;
            for (k=0 ; k<BPLength ; k++) {
               bit(1) Bz[i][j][k];
            }
         }
      }
   }
}
```

5.1.1.4.2 Semantics

NumTree: this is the number of trees, known as the number of edges in the base mesh.

LengthNbBits: this is the number of bits on which **BPLength** is read.

isPartial: this is a flag with value 0 if and only if the totality of the zerotrees are transmitted.

X, **Y**, **Z**: these are three flags with value 1 if bitplanes relative to the first, second and third components follow.

Ztld: this is the number of the currently sent or refined zero-tree.

BPLength: this is the length of the current bitplane.

component: this is a two-bits variable indicating which table (Bx, By or Bz) is updated.

FBP: this is the number of the first bitplane to read. Its value must be greater than or equal to 0 (most significant bitplane) and strictly lower than **NbBPX**, **NbBPY** or **NbBPZ**, depending on the component being read.

LBP: this is the number of the last bitplane to read. Its value must be greater than or equal to **FBP** and strictly lower than **NbBPX**, **NbBPY** or **NbBPZ**, depending on the component being read.

Bx, **By**, **Bz**: these are three 3-tables of bits used for encoding the first, second and third components of the wavelet coefficients. The first entry is the number of the zero-tree. The second one is the number of the current bitplane received. The third one is the number of the received bit in this plane.

NOTE – Decoding separately each component as independent zero-trees enables reception of normal meshes [40]. It also allows reception in different channels or decoding in separate threads.

The decoding process is exposed in Annex A.

5.1.2 Upstream syntax (for backchannel)

When specified as an upstream in corresponding ES descriptor, the WaveletSubdivisionSurface stream has to be read according to the AFX Generic Backchannel syntax (see Subclause 5.5).

5.2 MeshGrid stream

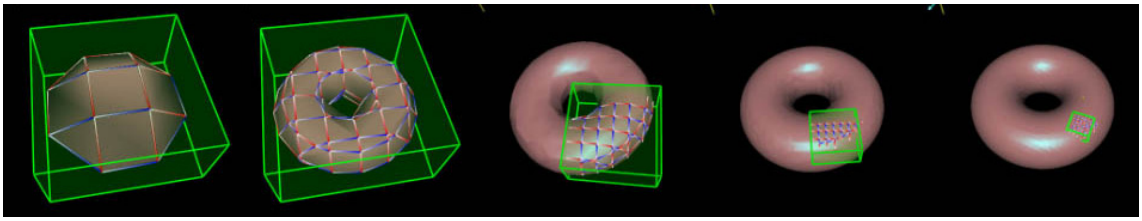
5.2.1 Overview

The MeshGrid stream has a modular structure consisting of a sequence of parts. There are several types of parts with different semantics, each one identified by a unique tag. All part types are optional, and several parts of the same type can be present in the stream, but their order is not imposed. The following parts of the MeshGrid stream are encoded at each resolution level, either as one single *region of interest* (ROI) or in separate ROIs, if view-dependent decoding is needed: (1) a *connectivity-wireframe description*, (2) a

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

reference-grid description, and (3) a *vertices-refinement description* (i.e. refining the position of the vertices relative to the reference-grid – the offsets).

Due to its regular nature, it is straightforward to divide the reference-grid into ROIs, and to encode the surface locally in each of these ROIs. As illustrated in Figure 29, single ROI mode can be mixed with multiple ROIs mode (view-dependent mode) for different resolution levels.



(a) (b) (c) (d) (e)

Figure 29 — Different resolution levels of a multi-resolution MeshGrid can be coded as one mesh (view-independent coding) or split into several ROI (view-dependent coding).

5.2.2 The Bitstream structure

The description of the stream is parsed in the *MeshGridDecoderConfig* class defined in subclause 5.2.2.2. The MeshGrid stream is parsed in the *MeshGridStream* class. The stream is organized as a succession of resolution level descriptions, starting from level 0 to (*totalNumLevelsMesh*-1), where *totalNumLevelsMesh* represents the total number of resolution levels of the mesh, and is computed in the MeshGridDecoderConfig class.

The mesh description part consists of a header, the *MeshDescriptor*, and a body consisting of three parts: the mesh connectivity coding (parsed in the *MeshConnectivity* class), the vertices' reposition bit plane coding (parsed in the *VertexRefine* class) and the vertices' refinement coding (parsed in the *VertexRefine* class). The grid description part consists of a header, parsed in the *GridDescriptor* class, and a body, parsed in the *GridCoefficients* class. In case the number of resolution levels of the grid (*totalNumLevelsGrid* defined in Annex B.1.2) is lower than the number of resolution levels of the mesh (*totalNumLevelsMesh*), the first *N* resolution level descriptions of the stream will only contain the mesh description part, with *N* defined as the difference between *totalNumLevelsMesh* and *totalNumLevelsGrid*. In case *totalNumLevelsGrid* is larger than *totalNumLevelsMesh*, the resolution '0' description of the stream contains one mesh description and $|N|$ grid descriptions.

The constituent parts in the MeshGrid stream must be decoded in the following order: for any resolution level of the mesh, the *MeshDescriptor* has to be parsed before retrieving any ROI for that resolution level. Decoding the ROIs implies decoding the mesh connectivity, which may be followed by parsing the vertices' reposition or refinement. Similarly, the grid decoding consists of parsing the header of the grid followed by the decoding the grid tiles.

5.2.2.1 Global Constants

```
const unsigned int LEVEL_BITS = 6;      Number of bits allocated for storing
                                        the number of resolution levels
const unsigned int REFINE_BITS = 6;     Number of vertices' refine bits
const unsigned int FILTER_BITS = 4;     For specifying the type of filter used
                                        for coding the grid.
const unsigned int ROI_BITS = 6;        Number of bits defined for a field
                                        specifying the length in bits allocated
                                        for the number of ROIs fields for each
                                        {u,v,w} direction (nROIs)
const unsigned int MIN_ROI_SIZE = 6;    The minimum ROI size is 5
```

© ISO/IEC 2006 – All rights reserved

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

`const unsigned int FIELD_BITS = 6;`	Number of bits for coding the $\log_2$ of the **threshold**. Used as well for storing the number of bits allocated for the scale values (**gridScale**) for each of the {x,y,z} wavelet coefficients.
`const unsigned int MIN_SCALE = 1;`	The minimum scale value is 1 (**gridScale**)
`const unsigned int QUANT_BITS = 6;`	The number of bits for storing the number of quantization bits of the {x,y,z} coordinates of the grid corners
`const unsigned int GRID_BITS = 5;`	Number of bits defined for a field specifying the length in bits allocated for the counters holding the size of the encoded wavelet coefficients for each {u,v,w} direction
`const unsigned int VERTS_BITS = 5;`	Number of bits defined for a field specifying the length in bits allocated for the counters holding the size of the encoded connectivity-wireframe and refine bits
`const unsigned int MIN_SPLIT_SIZE = 9;`	The minimum cube size that can be further split
`const unsigned int MIN_DEC_SIZE = 3;`	The minimum cube size that consists of several sub-bands
`const unsigned int GENERIC_MESH = 0`	The encoded mesh is generic, meaning that the triangulation may contain triangles, quadrilaterals, pentagons, hexagons and heptagons
`const unsigned int TRI_MESH = 1`	The encoded mesh contains only triangles
`const unsigned int QUADRI_MESH = 2`	The encoded mesh contains only quadrilaterals
`const unsigned int HEXA_MESH = 3`	The encoded mesh contains only hexagons
`const unsigned int READ_REPOSITION_BITS = 2`	No default value specified for the reposition bits

5.2.2.2 MeshGridDecoderConfig

5.2.2.2.1 Syntax

```
class MeshGridDecoderConfig extends AFXExtDescriptor: bit(8) tag=1{
   // the max number of resolution levels
   unsigned int totalNumLevelsMesh;

  // number of resolution levels specified for each u,v,w direction
  // of the reference-grid.
  FixedUVW nLevelsMesh(LEVEL_BITS);
  totalNumLevelsMesh = max(nLevelsMesh.u, max(nLevelsMesh.v, nLevelsMesh.w));

  // number of slices (reference surfaces) in the u,v,w directions
  // corresponding to the last resolution level
  ParsableUVW nSlices;

  // number of ROIs (regions of interest) in the u,v,w directions
  // corresponding to the last resolution level
  ParsableUVW nROIs;

  // flags
  bit (1) hasConnectivityInfo;
  bit (1) hasRefineInfo;
  bit (1) hasRepositionInfo;
  bit (1) hasGridInfo;

   // reserved
   bit(8) attributes = 0;

   // multilayer
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2006 – All rights reserved
Not for Resale

```
  bit (1) hasMultiLayer;
  if (hasMultiLayer) {
     ParseValue nLayersunsigned int (LAYER_BITS) nBitsLayer;
     unsigned int nBitsLayer = ceil(log2(nLayers)) + 1;
  }

  // type of homogeneous mesh
  unsigned int(2) meshType;

  if (meshType == QUADRI_MESH) {
     if (hasConnectivityInfo) {
     // connectivity bits and uniform quads splitting flags
     bit (1) sameBorderOrientation;
     bit (1) uniformSplit;
   }
   else {
     unsigned int sameBorderOrientation = 1;
     unsigned int uniformSplit = 1;
   }

   if (nSlices.w == 1)
      ParseValue offsetAmplitude;
  }

 // the choices for cyclic and folded mesh
 bit (3) cyclic_folded;

  // number of refine bits vertex
  RefineVertexDescriptor refine;

  // filter type for grid coding
  unsigned int(FILTER_BITS) filterType;

  // scale values for the x,y,z encoded grid coordinates
  // minimum scale factor is MIN_SCALE
  ScaleXYZ gridScale(FIELD_BITS);

  // the grid corners
  GridCorners gridCorner(nSlices, cyclic_folded);
}
```

5.2.2.2.2 Semantics

The **MeshGridDecoderConfig** class initializes the MeshGrid decoder. It **(1)** parses the resolution levels of the mesh (*nLevelsMesh*) for the $\{U,V,W\}$ directions with acceptable values for *nLevelsMesh.u, nLevelsMesh.v* and *nLevelsMesh.w,* lying in the range [1, 63], **(2)** computes the maximum number of resolution levels *totalNumLevelsMesh*, parses **(3)** the number of slices (*nSlices*) {S_U,S_V,S_W} and **(4)** the number of ROIs (*nROIs*) corresponding to the last resolution level of the reference-grid in the $\{U,V,W\}$ directions. The acceptable values for {*nSlices.u*, *nSlices.v, nSlices.w*} and {*nROIs.u*, *nROIs.v*, *nROIs.w*} lie in the range [1, 2^{63} -1]. Further it reads 6 flags defined in Table 9.

© ISO/IEC 2006 – All rights reserved

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

Table 9 — Meaning of the flags.

Flag	Meaning
hasConnectivityInfo	Boolean flag: 1. When set to '1' it indicates that the parts identified by the *MGMeshInfoTag*, *MGMeshConnectivityROInfoTag* and *MGMeshConnectivityInfoTag* tags can be present in the stream at any resolution level description. 2. When set to '0' it implies that the parts identified by the *MGMeshConnectivityROInfoTag* and *MGMeshConnectivityInfoTag* tags are not present in the stream. If the value of the *hasRefineInfo* flag is '1', then the part identified by the *MGMeshInfoTag* tag can be present in the stream only for the first resolution level description. If the *meshType* parameter defined in Table 10 has the value '2' then a default quadrilateral mesh is generated as explained in subclause 5.2.3.5.
hasRefineInfo	Boolean flag: 1. When set to '1' it indicates that the parts identified by the *MGMeshInfoTag*, *MGVerticesRefinementROIInfoTag* and *MGVerticesRefinementInfoTag* tags can be present in the stream at any resolution level description. If the value of the *hasConnectivityInfo* flag is '0' these parts can only be available for the first resolution level description, as explained in subclause 5.2.3.5. 2. When set to '0' it implies that the parts identified by the *MGVerticesRefinementROIInfoTag* and *MGVerticesRefinementInfoTag* tags are not present in the stream. The part identified by the *MGMeshInfoTag* tag can be present in the stream at any resolution level description only if the value of the *hasConnectivityInfo* flag is set to '1'.
hasRepositionInfo	Boolean flag: 1. It can be set to '1' only if the *hasConnectivityInfo* flag is also '1'. When equal to '1' it indicates that the parts identified by the *MGMeshInfoTag*, *MGVerticesRepositionROIInfoTag* and *MGVerticesRepositionInfoTag* tags can be present in the stream at any resolution level description except the last resolution level. 2. When set to '0' it implies that the parts identified by the *MGVerticesRepositionROIInfoTag* and *MGVerticesRepositionInfoTag* tags are not present in the stream. The part identified by the *MGMeshInfoTag* tag can be present in the stream at any resolution level description only if the value of one of the following two flags is set to '1': *hasConnectivityInfo*, *hasRefineInfo*.
hasGridInfo	Boolean flag: 1. When set to '1' it indicates that the parts identified by the *MGGridInfoTag*, *MGGridCoefficientsROIInfoTag* and *MGGridCoefficientsInfoTag* tags can be present in the stream at any resolution level description. 2. When set to '0' it implies that the parts identified by the *MGGridInfoTag*, *MGGridCoefficientsROIInfoTag* and *MGGridCoefficientsInfoTag* tags are not present in the stream. In this case the reference-grid points are uniformly distributed and their coordinates are computed as a linear interpolation between the eight grid corners parsed by *GridCorners* in the *MeshGridDecoderConfig* class.
attributes	A 8-bit flag reserved for future use.
hasMultiLayer	Boolean flag: 1. If set to '1' it indicates that the model consists of several surface layers. The identifiers for the surface layers are encoded on *nBitsLayer* bits. 2. When set to '0' then either the model consists of one surface layer or no distinction between the surface layers is made.
uniformSplit	Boolean flag: 1. When set to '1' it indicates that the stream contains a quadrilateral mesh allowing to obtain the connectivity-wireframe for the higher resolution levels by uniformly splitting each quad recursively into four sub-quads, as illustrated in Figure 45. The parts identified by the *MGMeshInfoTag*, *MGMeshConnectivityROInfoTag* and *MGMeshConnectivityInfoTag* tags are present in the stream only at the first resolution level description. 2. When set to '0' the parts identified by the *MGMeshInfoTag*, *MGMeshConnectivityROInfoTag* and *MGMeshConnectivityInfoTag* tags can be present in the stream at any resolution level description.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Further the **MeshGridDecoderConfig** class parses **(5)** the type of the mesh (*meshType*) as explained in Table 10.

Table 10 — Encoding of the mesh type (*meshType*)

Value	Meaning
0	GENERIC_MESH: Non-homogeneous mesh that may consist of polygons ranging from triangles to heptagons.
1	TRI_MESH: Homogeneous triangular mesh
2	QUADRI_MESH: Homogeneous quadrilateral mesh
3	HEXA_MESH: Homogeneous hexagonal mesh

If *meshType* is quadrilateral (*QUADRI_MESH*) than **(6)** a 1-bit flag (*sameBorderOrientation*) defines the number of bits used to store the connectivity links between the vertices, as explained in subclause 5.2.3.1.1, and defined in Table 11, and **(7)** another 1-bit flag (*uniformSplit*) indicates the type of multiresolution as explained in subclause 5.2.3.5 and Table 9. If the number of slices (*nSlices*) in any *W* direction is equal to '1', i.e. the reference-grid consists of one layer of points, then **(8)** the variable length variable *offsetAmplitude* defines the maximum value of the offsets as explained in subclause 4.3.4.3.2.

Table 11 — Encoding of the *sameBorderOrientation* flag

Value	Meaning
0	2-bits are used to encode a connectivity link between two vertices; it is the general case.
1	1-bit is used to encode a connectivity link between two vertices; it is a particular case which may occur for homogeneous quadrilateral meshes.

Further the **MeshGridDecoderConfig** class parses **(9)** the *cyclic_folded* mode variable defined in Table 12 that specifies the cyclic behaviour of the mesh, as explained in subclause 5.2.3.1.3, **(10)** the description of the vertices' refinement bits (*refine*), **(11)** the type of filter used in the wavelet transform (*filterType*), encoded on FILTER_TYPE bits, **(12)** the scaling factor for the {x,y,z} grid coefficients (*gridScale*), encoded on FIELD_BITS bits, **(13)** the {x,y,z} coordinates of the 8 corners of the reference-grid (*gridCorner*).

Table 12 — Encoding of the *cyclic_folded* mode variable.

Bits	Value	Meaning
000	0	NON_CYCLIC: general case non-cyclic non-folded mesh.
001	1	CYCLIC_U: cyclic mesh in the "U" direction.
011	3	CYCLIC UV: cyclic mesh in the "UV" direction.
101	5	FOLDED_SINGLE: cyclic mesh in the "U" direction and folded in the "V" direction at the end corresponding to the first index, i.e. $index = 0$. Notice that for the "V" direction the index varies between $index \in [0, nSlices[1] - 1]$.
111	7	FOLDED_BOTH: cyclic mesh in the "U" direction and folded in the "V" direction at both ends, i.e. $index = \{0, nSlices[1]\}$.
010	2	Reserved
100	4	Reserved
110	6	Reserved

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bits	Value	Meaning
000	0	NON_CYCLIC: general case non-cyclic non-folded mesh.
001	1	CYCLIC U: cyclic mesh in the "U" direction.
011	3	CYCLIC UV: cyclic mesh in the "UV" direction.
101	5	FOLDED_SINGLE: cyclic mesh in the "U" direction and folded in the "V" direction at the end corresponding to the first index, i.e. $index = 0$. Notice that for the "V" direction the index varies between $index \in [0, nSlices[1] - 1]$.
111	7	FOLDED_BOTH: cyclic mesh in the "U" direction and folded in the "V" direction at both ends, i.e. $index = \{0, nSlices[1]\}$.
010	2	Reserved
100	4	Reserved
110	6	Reserved

5.2.2.3 MeshGridStream

5.2.2.3.1 Syntax

```
aligned(8) expandable(2^28-1) class MeshGridStream
{
   MeshGridCommand[] commandUnits;
}
```

5.2.2.3.2 Semantics

The MeshGrid stream is an array of MeshGridCommand units.

5.2.2.4 MeshGridCommand

5.2.2.4.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class MeshGridCommand : bit(8) tag=0
{
   // empty. To be filled by classes extending this class.
}
```

5.2.2.4.2 Semantics

This is an abstract base class for the different types of command units of the MeshGrid stream. This class is extended by the classes identified by the class tags defined in Table 13.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 13 — MeshGrid command table

Tag value	Tag name	Description
0x00	Forbidden	
0x01	MGInfoTag	Tag for the MeshGrid stream coding information.
0x02	MGMeshInfoTag	Tag for the mesh coding information for a specified mesh resolution level.
0x03	MGGridInfoTag	Tag for grid coding information for a specified grid resolution level.
0x04	MGMeshConnectivityROIInfoTag	Tag for mesh connectivity information for a specified mesh resolution level and regions of interest (ROIs) list.
0x05	MGMeshConnectivityInfoTag	Tag for mesh connectivity information for a specified mesh resolution level.
0x06	MGVerticesRepositionROIInfoTag	Tag for vertices' reposition bits (single bit-plane) for a specified mesh resolution level and regions of interest (ROIs) list.
0x07	MGVerticesRepositionInfoTag	Tag for vertices' reposition bits (single bit-plane) for a specified mesh resolution level.
0x08	MGVerticesRefinementROIInfoTag	Tag for refinement bit-planes (the offset) for a specified mesh resolution level and regions of interest (ROIs) list.
0x09	MGVerticesRefinementInfoTag	Tag for refinement bit-planes (the offset) for a specified mesh resolution level.
0x10	MGGridCoefficientsROIInfoTag	Tag for wavelet coefficients for a specified grid resolution level and tiles list.
0x11	MGGridCoefficientsInfoTag	Tag for wavelet coefficients for a specified grid resolution level.
0x12	MGGridCornersInfoTag	Tag for the grid corners.
0x13-0xFE	Reserved for ISO use	
0xFF	Forbidden	

5.2.2.5 MGLevelDescriptor

5.2.2.5.1 Syntax

```
abstract class MGLevelDescriptor extends MeshGridCommand: bit(8) tag=0
{
   // read the variable length counter sizeOfInstance
   unsigned int(LEVEL_BITS) resolutionLevel;
   bit(4) flags;
}
```

5.2.2.5.2 Semantics

This is an abstract class that serves as a base class for the MeshGrid stream unit classes. MGLevelDescriptor reads **(1)** the resolution level (*resolutionLevel*) of the MeshGrid stream unit, and **(2)** a flag (*flags*) (defined in Table 11) that specifies how to handle the decoded MeshGrid stream unit. An example (exploiting the functionality provided by the *flags* field) showing the morphing of a MeshGrid model allowing for topological changes is displayed in Figure 30.

© ISO/IEC 2006 – All rights reserved

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

Table 14 — Values of *flags*

Bits	Meaning
0000	If first bit is '0' then the received data is an update to existing data.
0001	If first bit is '1' then the received data replaces any existing data.
	Remaining 3 bits are reserved.

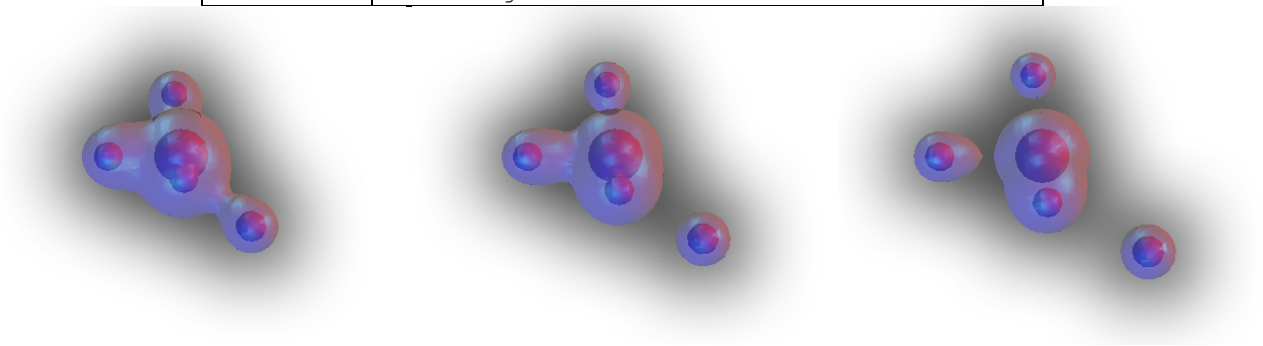


Figure 30 — Exploiting the functionality provided by the *flags* field for efficient morphing.

5.2.2.6 MGMeshDescriptor

5.2.2.6.1 Syntax

```
class MGMeshDescriptor(MeshGridDecoderConfig mgd) extends MGLayerDescriptor: bit(8)
tag=MGMeshInfoTag
{
bool bLastLevel = resolutionLevel.value == mgd.totalNumLevelsMesh - 1;
bool bReposition = mgd.hasRepositionInfo && !bLastLevel;
   bool bRefine = mgd.hasRefineInfo && (mgd.refine.bFull || bLastLevel);
   MeshDescriptor mdl[[resolutionLevel.value]](mgd.hasConnectivityInfo, bReposition,
bRefine);
}
```

5.2.2.6.2 Semantics

The MGMeshDescriptor class parses the mesh coding information for a specified mesh resolution level and layers.

5.2.2.7 MGLayerDescriptor

5.2.2.7.1 Syntax

```
abstract class MGLayerDescriptor extends MGLevelDescriptor(MeshGridDecoderConfig mgd):
bit(8) tag=0
{
   // read the variable length counter sizeOfInstance
   if (mgd.hasMultiLayer)
      ParseIndices identifier(mgd.nBitsLayer);
}
```

5.2.2.7.2 Semantics

This is an abstract class that serves as a base class for the MeshGrid stream unit classes. MGLayerDescriptor reads an *identifier* of the surface layer the MeshGrid stream unit refers to. When the value of *identifier.number* is '0' then the contents of the MeshGrid stream unit is generic for all surface layers. Figure 32 illustrates an example of a MeshGrid model consisting of several surface layers.

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

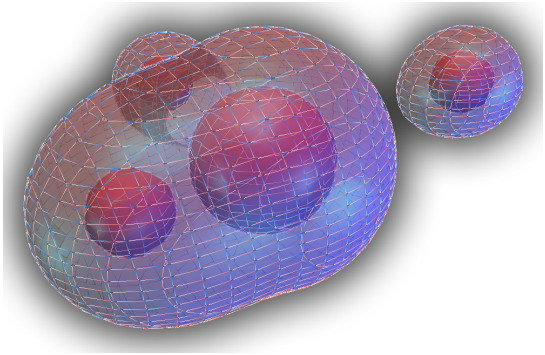


Figure 31 — MeshGrid model consisting of several surface layers.

5.2.2.8 MGGridCorners

5.2.2.8.1 Syntax

```
class MGGridCorners extends MeshGridCommand(MeshGridDecoderConfig mgd): bit(8)
tag=MGGridCornersInfoTag
{
   // read the coordinates of the grid corners
   GridCorners gridCorner(mgd.nSlices, mgd.cyclic_folded);
}
```

5.2.2.8.2 Semantics

The **MGGridCorners** class parses the grid corners (*gridCorner*) as explained in subclause 5.2.2.24.

5.2.2.9 MGGridDescriptor

5.2.2.9.1 Syntax

```
class MGGridDescriptor extends MGLevelDescriptor: bit(8) tag=MGGridInfoTag
{
   GridDescriptor gdl[[resolutionLevel.value]]();
}
```

5.2.2.9.2 Semantics

The MGGridDescriptor class parses the grid coding information for a specified grid resolution level.

5.2.2.10 MGMeshConnectivityROIDescriptor

5.2.2.10.1 Syntax

```
class MGMeshConnectivityROIDescriptor(MeshDescriptor mdl) extends MGLayerDescriptor:
bit(8) tag=MGMeshConnectivityROIInfoTag
{
   ParseIndices roi(mdl[resolutionLevel.value].nBitsIndex);
   MeshConnectivity
        mc[[resolutionLevel.value]](mdl[resolutionLevel.value].nBitsConnectivity,
                                              roi.number, roi.index);
}
```

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

© ISO/IEC 2006 – All rights reserved

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

5.2.2.10.2 Semantics

This class parses the mesh connectivity bits for specified layers, regions of interest (*roi*) and mesh resolution level (*resolutionLevel*).

5.2.2.11 MGMeshConnectivityDescriptor

5.2.2.11.1 Syntax

```
class MGMeshConnectivityDescriptor(MeshDescriptor mdl) extends MGLevelDescriptor: bit(8)
tag=MGMeshConnectivityInfoTag
{
   MakeIndices index(mdl[resolutionLevel.value].nROIs);
   MeshConnectivity mc[[resolutionLevel.value]]
(mdl[resolutionLevel.value].nBitsConnectivity,
                                                mdl[resolutionLevel.value].nROIs,
                                                index.value);
}
```

5.2.2.11.2 Semantics

This class parses the mesh connectivity bits for all layers, all regions of interest (identified by their indices (*index*)), and mesh resolution level (*resolutionLevel*).

5.2.2.12 MGVerticesRepositionROIDescriptor

5.2.2.12.1 Syntax

```
class MGVerticesRepositionROIDescriptor(MeshDescriptor mdl, MeshConnectivity mc) extends
MGLayerDescriptor: bit(8) tag=MGVerticesRepositionROIInfoTag
{
   ParseIndices roi(mdl[resolutionLevel.value].nBitsIndex);
   VertexRefine vrep[[resolutionLevel.value]](mdl[resolutionLevel.value].nBitsReposition,
                                              roi.number, roi.index,
                                              mc[resolutionLevel.value].bMeshPresent);
}
```

5.2.2.12.2 Semantics

This class parses the vertices' reposition bit-plane for specified layers, regions of interest (*roi*) and mesh resolution level (*resolutionLevel*).

5.2.2.13 MGVerticesRepositionDescriptor

5.2.2.13.1 Syntax

```
class MGVerticesRepositionDescriptor(MeshDescriptor mdl, MeshConnectivity mc) extends
MGLevelDescriptor: bit(8) tag=MGVerticesRepositionInfoTag
{
   MakeIndices index(mdl[resolutionLevel.value].nROIs);
   VertexRefine vrep[[resolutionLevel.value]](mdl[resolutionLevel.value].nBitsReposition,
                                              mdl[resolutionLevel.value].nROIs,
                      index.value,
                                              mc[resolutionLevel.value].bMeshPresent);
}
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.2.13.2 Semantics

This class parses the vertices' reposition bit-plane for all layers, all regions of interest (identified by their indices (*index*)), and mesh resolution level (*resolutionLevel*).

5.2.2.14 MGVerticesRefinementROIDescriptor

5.2.2.14.1 Syntax

```
class MGVerticesRefinementROIDescriptor(MeshDescriptor mdl, MeshConnectivity mc) extends
MGLayerDescriptor: bit(8) tag=MGVerticesRefinementROIInfoTag
{
   unsigned int(REFINE_BITS) startBitPlane;
   unsigned int(REFINE_BITS) endBitPlane;
   ParseIndices roi(mdl[resolutionLevel.value].nBitsIndex);
   VertexRefine vref[[resolutionLevel.value]](mdl[resolutionLevel.value].nBitsRefine,
                                              roi.number, roi.index,
                                              mc[resolutionLevel.value].bMeshPresent);
}
```

5.2.2.14.2 Semantics

This class parses the vertices' refinement bit-planes, starting with startBitPlane and ending with endBitPlane, for specified layers, regions of interest (*roi*) and mesh resolution level (*resolutionLevel*).

5.2.2.15 MGVerticesRefinementDescriptor

5.2.2.15.1 Syntax

```
class MGVerticesRefinementDescriptor(MeshDescriptor mdl, MeshConnectivity mc) extends
MGLevelDescriptor: bit(8) tag=MGVerticesRefinementInfoTag
{
   MakeIndices index(mdl[resolutionLevel.value].nROIs);
   VertexRefine vref[[resolutionLevel.value]](mdl[resolutionLevel.value].nBitsRefine,
                                              mdl[resolutionLevel.value].nROIs,
                     index.value,
                                              mc[resolutionLevel.value].bMeshPresent);
}
```

5.2.2.15.2 Semantics

This class parses all the vertices' refinement bit-planes for all layers, all regions of interest (identified by their indices (*index*)), and mesh resolution level (*resolutionLevel*).

5.2.2.16 MGGridCoefficientsROIDescriptor

5.2.2.16.1 Syntax

```
class MGGridCoefficientsROIDescriptor(GridDescriptor gdl) extends MGLevelDescriptor:
bit(8) tag=MGGridCoefficientsROIInfoTag
{
   unsigned int(FIELD_BITS) startBitPlane;
   ParseIndices tile(gdl[resolutionLevel.value].nBitsIndex);
   GridCoefficients gc[[resolutionLevel.value]] (gdl[resolutionLevel.value].counter,
                                                 gdl[resolutionLevel.value].nBitsCounter,
                                                 tile.number, tile.index);
}
```

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

© ISO/IEC 2006 – All rights reserved

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

5.2.2.16.2 Semantics

This class parses parts of the component streams, i.e. three binary streams each of them corresponding to one of the {*x*,*y*,*z*} wavelet encoded coordinates of the grid points, for specified tiles (*tile*) and grid resolution level (*resolutionLevel*). The start bitplane (*startBitPlane*) is specified and it is the same for each component stream. The number of bytes received for each component stream and the corresponding wavelet coefficients are parsed by the *GridCoefficients* class.

5.2.2.17 MGGridCoefficientsDescriptor

5.2.2.17.1 Syntax

```
class MGGridCoefficientsDescriptor(GridDescriptor gdl) extends MGLevelDescriptor: bit(8)
tag=MGGridCoefficientsInfoTag
{
   MakeIndices index(gdl[resolutionLevel.value].totalNumTiles);
   GridCoefficients gc[[resolutionLevel.value]] (gdl[resolutionLevel.value].counter,
                                                 gdl[resolutionLevel.value].nBitsCounter,
                                                 gdl[resolutionLevel.value].totalNumTiles,
                                                 index.value);
}
```

5.2.2.17.2 Semantics

This class parses parts of the component streams, i.e. three binary streams each of them corresponding to one of the {*x*,*y*,*z*} wavelet encoded coordinates of the grid points, for all the tiles,identified by their indices (*index*),and grid resolution level (*resolutionLevel*). The number of bytes received for each component stream and the corresponding wavelet coefficients are parsed by the *GridCoefficients* class.

5.2.2.18 MeshDescriptor

5.2.2.18.1 Syntax

```
// Mesh Description for level
aligned(8) class MeshDescriptor(bool hasConnectivity, bool hasReposition, bool hasRefine)
{
   // some member variables
   unsigned int nBitsIndex, totalNumROIs;

      // compute the number of slices and number of ROIs at resolutionLevel
(MGLevelDescriptor)
   // from nROIs, nLevels and nSlices specified in MeshGridDecoderConfig (mgd)
   ComputeNrSlices nSlices(mgd.nSlices, mgd.nLevels, resolutionLevel);
   ComputeNrROIs nROIs(mgd.nROIs, mgd.nLevels, resolutionLevel);

 if (hasReposition) {
     // retrieve the reposition bits flag
     bit (2) repositionBits;
 }

   if (hasConnectivity) {
      // number of bits for the ROI indices fields
      nBitsIndex = (int) floor(log2(nROIs.totalNumROIs) + 1);
      // number of bits for the counter fields
      unsigned int(VERTS_BITS) nBitsConnectivity;
   }

   if (hasReposition && repositionBits == READ_REPOSITION_BITS) {
      unsigned int(VERTS_BITS) nBitsReposition;
   }
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if (hasRefine) {
      unsigned int(VERTS_BITS) nBitsRefine;
   }
}
```

5.2.2.18.2 Semantics

The **MeshDescriptor** class parses the coding information for a specified mesh resolution level. It first computes **(1)** the number of slices (*nSlices*) and **(2)** the number of ROIs (*nROIs*) for each {*u,v,w*} direction. Further, if the *hasReposition* flag is set to '1' (the *hasRepositionInfo* flag from the *MeshGridDecoderConfig* is set to '1' and the specified mesh resolution level is not the last) it reads **(3)** a 2-bit flag (*repositionBits*), which indicates the default value of the vertices' reposition bits or their presence in the stream as given in Table 15. When a default value is specified, the reposition bits are not present in the stream for the specified resolution level.

Table 15 — Encoding of the *repositionBits* flag

Value	Meaning
0	No reposition bits encoded, all have default value 0.
1	No reposition bits encoded, all have default value 1.
2	No default value, the reposition bits are present in the stream.
3	Reserved

Further, if the *hasConnectivity* flag is set to '1' (the *hasConnectivityInfo* flag from the *MeshGridDecoderConfig* is set to '1') the **MeshDescriptor** class **(4)** computes the number of bits (*nBitsIndex*) in which the indices of the ROIs are stored, and **(5)** parses the number of bits (*nBitsConnectivity*) used for storing the length (in bytes) of the coded mesh connectivity. If the *hasReposition* flag is set to '1' and no default value is specified for the vertices' reposition bits (*repositionBits*), then **(6)** the number of bits (*nBitsReposition*) used for storing the length (in bytes) of the coded vertices' reposition information is retrieved. If the *hasRefine* flag is '1' (the *hasRefineInfo* flag from the *MeshGridDecoderConfig* is set to '1' and it is the last resolution level or the full refine flag has been specified), then **(7)** the number of bits (*nBitsRefine*) used for storing the length (in bytes) of the coded vertices' refinement information is parsed.

5.2.2.19 GridDescriptor

5.2.2.19.1 Syntax

```
// Grid Description for level
aligned(8) class GridDescriptor() {
   // some member variables
   unsigned int nBitsIndex, totalNumTiles;

   // compute the number of slices and Tiles at resolutionLevel (MGLevelDescriptor) from
   // nROIs, nLevels and nSlices specified in MeshGridDecoderConfig (mgd)
   ComputeNrSlices nSlices(mgd.nSlices, mgd.nLevels, resolutionLevel);
   ComputeNrROIs nTiles(mgd.nROIs, mgd.nLevels, resolutionLevel);

   // number of bits for the ROI indices fields
   nBitsIndex = (int) floor(log2(nTiles.totalNumROIs) + 1);

   // the threshold
   Threshold threshold;

   // allocate an array keeping the number of bytes per Tile
   // it will be initialized in GridCoefficients
   PointXYZ counter[nTiles.totalNumROIs];
   for (i = 0; i < nTiles.totalNumROIs; i++) {
      counter[i] = {0,0,0};
   }

   // number of bits for the counter fields (it can be 0)
   ParsableXYZ nBitsCounter(GRID_BITS);
}
```

© ISO/IEC 2006 – All rights reserved

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

5.2.2.19.2 Semantics

The **GridDescriptor** class parses the grid header for a given resolution level. It first computes **(1)** the number of slices (*nSlices*) and **(2)** the number of tiles (*nTiles*) for each {*u,v,w*} direction. Further, it **(3)** parses the threshold (*threshold*) for each {x,y,z} coded grid tile, **(4)** computes the number of bits needed for specifying the tile indices, **(5)** allocates the array which stores the number of bytes of coefficients received per tile, and **(6)** parses a GRID_BITS bits value (*nBitsCounter*), representing the number of bits allocated for the counters storing the length (in bytes) for each {x,y,z} coded grid tile.

5.2.2.20 MeshConnectivity

5.2.2.20.1 Syntax

```
class MeshConnectivity (unsigned int nBitsCounter, unsigned int numberOfROIs, unsigned int
index[])
{
   unsigned int i = 0;

   // get ROIs lookup
   for (i = 0; i < numberOfROIs; i++) {
      bit(1) bMeshPresent[[index[i]]];
   }

   // get the counters for the ROIs
   for (i = 0; i < numberOfROIs; i++) {
      int counter[[index[i]]] = 0;
      if (bMeshPresent[index[i]] == 1) {
         int (nBitsCounter) counter[[index[i]]];
      }
   }

   // get the ROI coding bits
   for (i = 0; i < numberOfROIs; i++) {
      unsigned int count;
      for (count = 0; count < counter[index[i]]; count++) {
         unsigned int(8) data;
      }
   }
}
```

5.2.2.20.2 Semantics

The **MeshConnectivity** class parses the mesh connectivity data for a set of ROIs at a specified resolution level. It reads, for each ROI (identified by the *index*), a 1-bit flag (*bMeshPresent*), which, if true, means that the mesh is passing through the ROI. Further, if *bMeshPresent* is true the **MeshConnectivity** class reads *nBitsCounter* bits representing the length (in bytes), specified by the *counter*, of the coded connectivity information. The connectivity information is stored in buffer *data*. The decoding of the connectivity information is explained in subclause 5.2.3.1.

5.2.2.21 VertexRefine

5.2.2.21.1 Syntax

```
class VertexRefine (unsigned int nBitsCounter, unsigned int numberOfROIs, unsigned int
index[], bit bMeshPresent[])
{
   unsigned int i = 0;

   // get the counters for the ROIs
   for (i = 0; i < numberOfROIs; i++) {
      int counter[[index[i]]] = 0;
      if (bMeshPresent[index[i]] == 1) {
         int (nBitsCounter) counter[[index[i]]];
      }
   }
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   // get the vertices' reposition/refine bits
   for (i = 0; i < numberOfROIs; i++) {
      unsigned int count;
      for (count = 0; count < counter[index[i]]; count++) {
         unsigned int(8) data;
      }
   }
}
```

5.2.2.21.2 Semantics

The **VertexRefine** class parses the vertices refine/reposition data for a set of ROIs at a specified resolution level. It checks if the mesh is present in the ROI (identified by the *index*) by testing the *bMeshPresent* flag, which is initialized by the *MeshConnectivity* class. Further, if the *bMeshPresent* evaluates to true, the **VertexRefine** class reads nBitsCounter bits representing the length (in bytes) of the coded reposition/refinement information. The reposition/refinement information is stored in buffer *data*. The decoding of the reposition/refinement information is explained in subclause 5.2.3.3.

5.2.2.22 GridCoefficients

5.2.2.22.1 Syntax

```
class GridCoefficients(PointXYZ counter[], ParsableXYZ nBitsGrid, unsigned int
numberOfTiles, unsigned int index[])
{
   unsigned int i = 0;

   // get the counters for the tiles and update the total counters
   // in case the grid coefficients are read progressively
   for (i = 0; i < numberOfTiles; i++) {
      CounterXYZ frameCounter[[index[i]]](nBitsGrid);
      counter[index[i]] += frameCounter[index[i]];
   }

   // read the coefficients and append them to the existing ones
   // in case the grid coefficients are read progressively
   for (i = 0; i < numberOfTiles; i++) {
      unsigned int count;

      // read the coded coefficients of the 'x' coordinates
      for (count = 0; count < frameCounter[index[i]].x; count++)
         unsigned int(8) dataX;

      // read the coded coefficients of the 'y' coordinates
      for (count = 0; count < frameCounter[index[i]].y; count++)
         unsigned int(8) dataY;

      // read the coded coefficients of the 'z' coordinates
      for (count = 0; count < frameCounter[index[i]].z; count++)
         unsigned int(8) dataZ;
   }
}
```

5.2.2.22.2 Semantics

The **GridCoefficients** class parses the grid coefficients data for a set of ROIs at a specified resolution level. It reads for each tile (identified by the *index*) the length (in bytes), specified in *frameCounter* for each coordinate {x,y,z}, from the coded grid information. The decoding of the grid information is explained in subclause 5.2.3.2. The grid coefficients are stored in the buffers *dataX*, *dataY*, *dataZ*.

© ISO/IEC 2006 – All rights reserved

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

5.2.2.23 RefineVertexDescriptor

5.2.2.23.1 Syntax

```
// number of refine bits vertex
class RefineVertexDescriptor {
   if (hasRefineInfo) {
      // full refine at each level
      bit(1)  bFull;

      // number of refine bits should be larger than 0
      // otherwise the hasRefineInfo flag should be 0
      unsigned int(REFINE_BITS) nBits;
   }
   else {
      unsigned int bFull = 0;
      unsigned int nBits = 0;
   }
}
```

5.2.2.23.2 Semantics

The **RefineVertexDescriptor** class reads **(1)** the 1-bit flag (*bFull*) indicating the presence of vertices' refinement information coded for each resolution level of the mesh, and **(2)** REFINE_BITS bits value representing the number of quantization bits of the vertices' offsets (*nBits*).

5.2.2.24 GridCorners

5.2.2.24.1 Syntax

```
class GridCorners(PointUVW nSlices, BYTE cyclic_folded) {
  unsigned int(QUANT_BITS) nBits;

   // check for single layer and cyclic/folded
   unsigned int numCorners = 8;
   if (nSlices.w == 1)
      numCorners /= 2;
   if (cyclic_folded == CYCLIC_U)
      numCorners /= 2;
   else if (cyclic_folded == CYCLIC_UV)
      numCorners /= 4;

   unsigned int idx;
   PointXYZ value[numCorners];
   for (idx = 0; idx < numCorners; idx ++) {
      // the x coordinate
      bit(1) sgn;                 // the sign
      unsigned int(nBits) tmp;    // the value

      if (sgn == 1)   value[idx].x = -tmp;
      else            value[idx].x =  tmp;

      // the y coordinate
      bit(1) sgn;                 // the sign
      unsigned int(nBits) tmp;    // the value

      if (sgn == 1)   value[idx].y = -tmp;
      else            value[idx].y =  tmp;

      // the z coordinate
      bit(1) sgn;                 // the sign
      unsigned int(nBits) tmp;    // the value
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      if (sgn == 1)   value[idx].z = -tmp;
      else            value[idx].z =  tmp;
   }
}
```

5.2.2.24.2 Semantics

The **GridCorners** class retrieves the coordinates {x,y,z} of the eight corners of the grid. It reads **(1)** a QUANT_BITS bits value (*nBits*) indicating the number of bits allocated for the *value* fields. For each of the {x,y,z} coordinates of the corners**,** it reads **(2)** a 1-bit *sgn*, which, when equal to '1', indicates that the following value is negative, and **(3)** a *nBit* value (*value*), representing the absolute coordinate value.

5.2.2.25 ScaleXYZ

5.2.2.25.1 Syntax

```
// group the x,y,z directions
class ScaleXYZ(unsigned int count) {
  // count and nBits should be larger than 0
  unsigned int(count) nBits;

  // the actual fields
  unsigned int(nBits) x;
  unsigned int(nBits) y;
  unsigned int(nBits) z;
}
```

5.2.2.25.2 Semantics

The **ScaleXYZ** class parses the scaling factors for decoding the wavelet coefficients of the {x,y,z} coordinates of the reference-grid points. It reads *nBits* bits for each coordinate {*x,y,z*}. The scaling values are used to scale down the decoded wavelet coefficients by means of integer division, before applying the wavelet reconstruction (see subclause 5.2.3.2.1) of the reference-grid coordinates.

5.2.2.26 PointXYZ

5.2.2.26.1 Syntax

```
class PointXYZ {
  // the actual fields
  int x, y, z;
}
```

5.2.2.26.2 Semantics

The **PointXYZ** class is a structure grouping the coordinates {x,y,z} of the reference-grid points. It does not read values from the stream.

5.2.2.27 ParsableUVW

5.2.2.27.1 Syntax

```
class ParsableUVW() {
   unsigned int u = ParseValue();
   unsigned int v = ParseValue();
   unsigned int w = ParseValue();
}
```

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

© ISO/IEC 2006 – All rights reserved

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

5.2.2.27.2 Semantics

The **ParsableUVW** class reads three variables $\{u,v,w\}$ that are related to the $\{U,V,W\}$ directions of the reference-grid.

5.2.2.28 PointUVW

5.2.2.28.1 Syntax

```
class PointUVW {
  // the actual fields
  unsigned int u, v, w;
}
```

5.2.2.28.2 Semantics

The **PointUVW** class is a structure grouping the positions $\{u,v,w\}$ in the reference-grid. It does not read values from the stream.

5.2.2.29 FixedUVW

5.2.2.29.1 Syntax

```
// group the u,v,w directions
class FixedUVW (unsigned int nBits) {
  unsigned int(nBits) u;
  unsigned int(nBits) v;
  unsigned int(nBits) w;
}
```

5.2.2.29.2 Semantics

The **FixedUVW** class reads three variables $\{u,v,w\}$ that are related to the $\{U,V,W\}$ directions of the reference-grid.

5.2.2.30 ParsableXYZ

5.2.2.30.1 Syntax

```
class ParsableXYZ(unsigned int nBits) {
   unsigned int(nBits) x;
   unsigned int(nBits) y;
   unsigned int(nBits) z;
}
```

5.2.2.30.2 Semantics

The **ParsableXYZ** class reads three variables $\{x,y,z\}$ that are related to the coordinates of the reference-grid.

5.2.2.31 CounterXYZ

5.2.2.31.1 Syntax

```
// group the x,y,z coordinates
class CounterXYZ(ParsableXYZ nBits) {
  // the actual fields
  unsigned int x, y, z;
  x = y = z = 0;
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  // number of bits allocated for the x,y,z fields; it can be '0'
  if (nBits.x > 0) {
    // coding of the x coordinate
    bit(1) bGridCoded;
    if (bGridCoded == 1) unsigned int(nBits.x) x;
  }

  if (nBits.y > 0) {
    // coding of the y coordinate
    bit(1) bGridCoded;
    if (bGridCoded == 1) unsigned int(nBits.y) y;
  }

  if (nBits.z > 0) {
    // coding of the z coordinate
    bit(1) bGridCoded;
    if (bGridCoded == 1) unsigned int(nBits.z) z;
  }
}
```

5.2.2.31.2 Semantics

The **CounterXYZ** class reads the counters storing the length (in bytes) of the coded grid tiles for each of the {x,y,z} grid coordinates. If the *nBits* value is equal to '0', the counters {*x,y,z*} are set to '0'; otherwise, for each coordinate, the CounterXYZ class reads **(1)** the 1-bit flag (*bGridCoded*), indicating, when '0', that no grid coefficients have been coded, thus no counter ({*x,y,z*}) is present; otherwise, the class retrieves the **(2)** *nBits* bits counters ({*x,y,z*}) storing the length (in bytes) of the coded grid tiles.

5.2.2.32 Threshold

5.2.2.32.1 Syntax

```
class Threshold {
  // the actual fields
  unsigned int x, y, z;
  x = y = z = 0;

  // the threshold for the coded x coordinate
  unsigned int(FIELD_BITS) t;
  if (t != 0)   x = 1 << (t - 1);

  // the threshold for the coded y coordinate
  unsigned int(FIELD_BITS) t;
  if (t != 0)   y = 1 << (t - 1);

  // the threshold for the coded z coordinate
  unsigned int(FIELD_BITS) t;
  if (t != 0)   z = 1 << (t - 1);
}
```

5.2.2.32.2 Semantics

For each of the {x,y,z} coded coordinates, the **Threshold** class reads, for each sub-band, the $\log_2$ of the threshold value (*t*). If *t* is equal to '0', the corresponding threshold {*x,y,z*} is '0'. Otherwise, the threshold is computed as $2^{(t-1)}$ (see subclause 5.2.3.2.2).

© ISO/IEC 2006 – All rights reserved

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

5.2.2.33 ParseIndices

5.2.2.33.1 Syntax

```
class ParseIndices(int nBitsIndex)
{
   unsigned int i = 0, j = 0, k = 0;

   // the number of elements in the indices list.
   unsigned int (nBitsIndex) number;

   for (i = 0; i < number; i++) {
      bit (1) isIndexed;
      if (isIndexed) {
         unsigned int (nBitsIndex) index[[k++]]; // index of the unit
      } else {
         unsigned int (nBitsIndex) start;        // start index of the unit
         unsigned int (nBitsIndex) count;        // number of indices in the range
         for (j = 0; j < count; j++) {
            index[[k++]] = start + j;
         }
      }
   }

   // assign the total number of units.
   number = k;
}
```

5.2.2.33.2 Semantics

The **ParseIndices** class retrieves a list of units that can be independently read, given a set of explicitly defined indices, and/or units that can be successively read, given a range of consecutive indices. First, it reads the number of elements in the indices list (*number*). A list element is the index of a unit (*index*), if the flag *isIndexed* is set to '1'; otherwise, the **ParseIndices** class retrieves the start index (*start*) of a range of indices, followed by a number (*count*) specifying the number of indices in the range. The indices are stored in *index* array.

5.2.2.34 ParseValue

5.2.2.34.1 Syntax

```
class ParseValue
{
   unsigned int length = 0;
   unsigned int number = 0;

   do {
      bit(1) bNextByte;
      number <<= 7;
      bit(7) value;
      number |= value;
      length += 8;
   }
   while (bNextByte);
}
```

5.2.2.34.2 Semantics

The **ParseValue** class parses a variable length integer. The result is returned in *number* and the number of bytes read in *length*.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.2.35 MakeIndices

5.2.2.35.1 Syntax

```
class MakeIndices (int number)
{
   unsigned int i = 0;
   unsigned int value[number];

   for (i = 0; i < number; i++) {
      value[i] = i;
   }
}
```

5.2.2.35.2 Semantics

The **MakeIndices** class creates a list of consecutive indices. There is no stream access in this class.

5.2.2.36 MGDescriptor

5.2.2.36.1 Syntax

```
class MGDescriptor extends MeshGridCommand: bit(8) tag=MGInfoTag
{
   // read the variable length counter sizeOfInstance
   MeshGridDecoderConfig decoderConfig;
}
```

5.2.2.36.2 Semantics

The MGDescriptor class parses the coding information of the MeshGrid stream. This part is mandatory to be present at the beginning of the stream when the MeshGrid stream is carried in the buffer field of the Bitwrapper node during the in-band scenario, as explained in ISO/IEC 14496-11.

5.2.2.37 ComputeNrROIs

5.2.2.37.1 Syntax

```
class ComputeNrROIs (PointUVW nMaxROIs, PointUVW nLevels, unsigned int resolutionLevel)
{
   // compute the total number of resolution levels (totalNumLevels) and
   // for each {U,V,W} direction find the difference between totalNumLevels
   // and current resolution level (resolutionLevel)
   unsigned int totalNumLevels = max(nLevels.u, max(nLevels.v, nLevels.w));
   PointUVW levelDiff = min(totalNumLevels – resolutionLevel – 1, nLevels - 1);

   // compute the nROIs in the {U,V,W} directions
   unsigned int u = DevideByTwoEven(nMaxROIs.u, levelDiff.u);
   unsigned int v = DevideByTwoEven(nMaxROIs.v, levelDiff.v);
   unsigned int w = DevideByTwoEven(nMaxROIs.w, levelDiff.w);

   unsigned int totalNumROIs = u * v * w;
}
```

5.2.2.37.2 Semantics

The **ComputeNrROIs** class computes the number of ROIs of a lower resolution level given the number of ROIs (*nROIs*) at a higher resolution level and the level difference (*levelDiff*). There is no stream access in this class.

© ISO/IEC 2006 – All rights reserved

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

5.2.2.38 ComputeNrSlices

5.2.2.38.1 Syntax

```
class ComputeNrSlices (PointUVW nSlices, PointUVW nLevels, unsigned int resolutionLevel)
{
   // compute the total number of resolution levels (totalNumLevels) and
   // for each {U,V,W} direction find the difference between totalNumLevels
   // and current resolution level (resolutionLevel)
   unsigned int totalNumLevels = max(nLevels.u, max(nLevels.v, nLevels.w));
   PointUVW levelDiff = min(totalNumLevels – resolutionLevel – 1, nLevels - 1);

   unsigned int u = DevideByTwoOdd(nSlices.u, levelDiff.u);
   unsigned int u = DevideByTwoOdd(nSlices.v, levelDiff.v);
   unsigned int w = DevideByTwoOdd(nSlices.w, levelDiff.w);
}
```

5.2.2.38.2 Semantics

The **ComputeNrSlices** class computes the number of slices of a lower resolution level given the number of slices (*nSlices*) at a higher resolution level and the level difference (*levelDiff*). There is no stream access in this class.

5.2.2.39 DevideByTwoEven

5.2.2.39.1 Syntax

```
class DevideByTwoEven (int number, int times)
{
   unsigned int val = (number >> times);
}
```

5.2.2.39.2 Semantics

The **DevideByTwoEven** class performs the division of an even *number* by two to power *times*. There is no stream access in this class.

5.2.2.40 DevideByTwoOdd

5.2.2.40.1 Syntax

```
class DevideByTwoOdd (int number, int times)
{
   unsigned int val = ((number + (1 << times) – 1) >> times);
}
```

5.2.2.40.2 Semantics

The **DevideByTwoOdd** class performs the division of an odd *number* by two to power *times*. There is no stream access in this class.

5.2.3 Decoder

The *connectivity-wireframe* is coded using a new type of 3D extension of Freeman chain-code. The *reference-grid* is a vector field defined on a regular discrete 3D space, and the coordinates ($x(u,v,w)$,$y(u,v,w)$,$z(u,v,w)$), are compressed using an embedded 3D wavelet-based multi-resolution intra-band coding algorithm.

The decoder of the MeshGrid bitstream consists of three parts: (1) the connectivity-wireframe decoder, (2) the reference-grid decoder, and (3) the vertices' refinement decoder.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.3.1 Connectivity-Wireframe Decoder

The connectivity-wireframe description is stored in the ***vertexLink*** field of the **MeshGrid** node (see subclause 4.3.4.2.2), and defines the connectivity vectors, between the vertices of the mesh as illustrated in Figure 32 for one reference-surface through a 3D object, and for the entire 3D object in Figure 33. A connectivity vector (label 7) has three constraints: (1) given a starting vertex V_P, a connectivity vector -*L*- from V_P will be located inside one of the reference-surfaces S_1 or S_2 passing through V_P, (2) a connectivity-vector will connect two vertices V_P and V_N that are lying the closest to each other at the same side of the object's surface, (3) the orientation of a connectivity vector inside reference-surface *S*, e.g. from vertex V_P to V_N, is defined by a counter-clockwise (*CCW*) or a clockwise (*CW*) scanning direction around a central point inside the object. The *CCW* scanning direction is imposed when the vertices V_P and V_N linked by the connectivity vector, are located on the external surface of the object. Respectively, the *CW* scanning direction is imposed when the vertices V_P and V_N are located on the internal surface of the object.

There will be two connectivity-vectors (-L_{N1}- and -L_{N2}-) going from vertex *V* to other vertices (*outgoing*), one connectivity vector being located inside reference-surface S_1, while the other one is located inside reference-surface S_2. Similarly, vertex *V* will be referred to by two *incoming* links (-L_{P1}- and -L_{P2}-) from two other vertices. The 4 neighboring vertices with *V* are named P_1, P_2, N_1 and N_2 (see Figure 33), and are connected with *V* via -L_{P1}-, -L_{P2}-, -L_{N1}- and -L_{N2}-. Vertex N_1 (respectively N_2) follows vertex *V* on curve C_1 (respectively C_2), and vertex P_1 (respectively P_2) precedes vertex *V* in curve C_1 (respectively C_2) for the imposed scanning orientation.

© ISO/IEC 2006 – All rights reserved

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

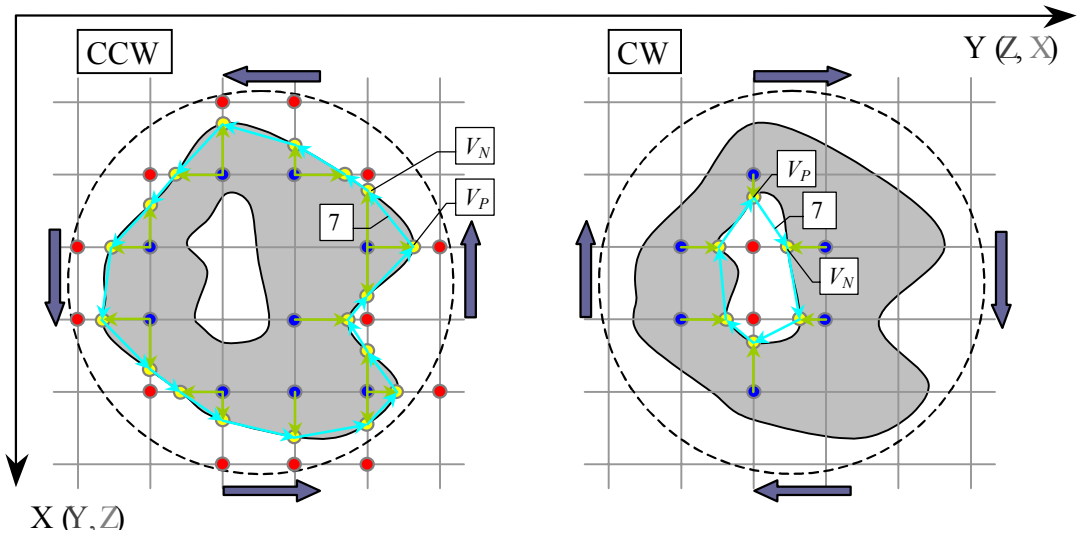


Figure 32 — A cross-section through a 3D object, illustrating the scanning direction, which is CCW for an external surface and CW for an internal surface. The connectivity vectors have the orientation of the scanning direction.

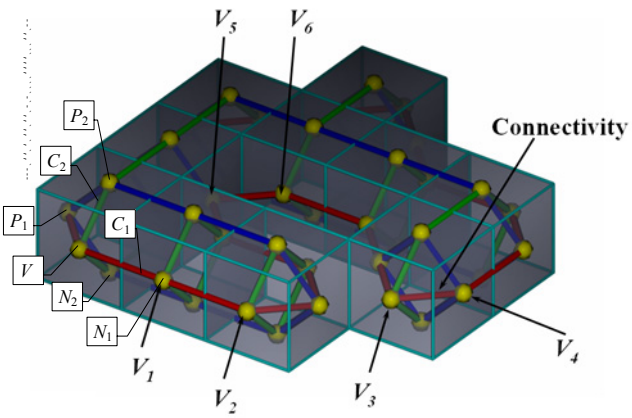


Figure 33 — The connectivity-wireframe of a discrete object, and the corresponding border voxels. It illustrates the relative positions between two connected vertices.

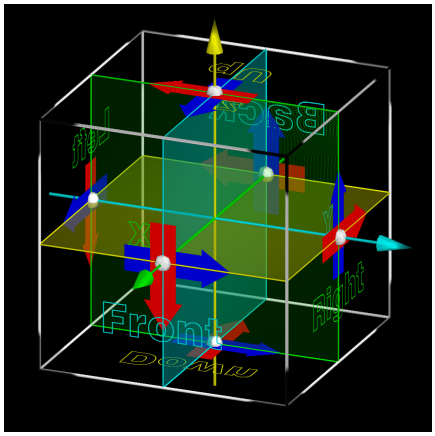


Figure 34 — The 6 discrete orientations of the faces of the border voxel, labeled as: Front, Back, Left, Right, Up, Down.

The indices {1, 2} of the connectivity vectors (-L_1-) and (-L_2-) are chosen in such a way that the cross product defined by Equation 15 returns a normal vector oriented outwards the surface of the object at the position of vertex *V*.

$$N = (L_{N1} - L_{P1}) \times (L_{N2} - L_{P2})$$

Equation 15

According to Equation 15, Table 17 illustrates the correct identification of the -L_1- and -L_2- connectivity vectors attached to vertices located on one of the 6 discrete border faces (see Figure 34), given the imposed scanning orientation for a connectivity path *C*: i.e. *CCW* for external, respectively *CW* for internal surfaces. In order to satisfy Equation 15 the connectivity vectors of two consecutive vertices may change their ordering ($L_1 \rightarrow L_2$ and $L_2 \rightarrow L_1$). The exclamation marks from Table 17 indicate cases that are not possible.

The connectivity-wireframe has been coded using a new type of Freeman chain code extended to 3D, which is a typical coding method for the discrete space. For the MeshGrid representation, the discrete space is represented by the discrete (*u*,*v*,*w*) positions of the reference-grid to which the vertices are attached to.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The relative discrete border direction of two consecutive vertices along any scanning direction C_1 or C_2 (see Figure 33) may have only three different orientations, and therefore are coded on two bits only, having the meaning shown in Table 16 (b).

Table 16 — Encoding of the relative border direction (orientation) of two consecutive vertices in (a) for the particular case when the vertices have the same discrete border direction, and in (b) the general case. The encoding of the discrete border direction of the starting vertex is given in (c)

Bits	Meaning
0	same discrete border orientation – case V_1 and V_2
1	broken connectivity (open meshes)

(a)

Bits	Meaning
00	same discrete border orientation – case V_1 and V_2
01	consecutive discrete border directions are rotated 90° *CCW* – case V_3 and V_4
10	consecutive discrete border directions are rotated 90° *CW* – case V_5 and V_6
11	broken connectivity (open meshes)

(b)

Bits	Meaning
000	reserved
001	Back face
010	Front face
011	Left face
100	Right face
101	Bottom face
111	Top face

(c)

A fourth value is added to indicate broken connectivities in open meshes.

Yet, in the particular case when the mesh is homogeneous quadrilateral, which is defined when variable *meshType*, initialized in the MeshGridDecoderConfig class as explained in subclause 5.2.2.2, yields the value QUADRI_MESH, and the *sameBorderOrientation* flag is set, then all the vertices are located on the same discrete border orientation with respect to the {*u*,*v*,*w*} direction of the reference-grid. In this case 1 bit suffices to encode the connectivity between two consecutive vertices, as illustrated in Table 16 (a).

Further, Table 17 illustrates the relation between the discrete position of two connected vertices and their discrete border direction.

Although there are four connectivity-vectors for each boundary vertex in the connectivity-wireframe, only the outgoing connectivity vectors (-L_{N1}- and -L_{N2}-) need to be present in the stream.

The decoding of the connectivity-wireframe requires that the starting vertex V_S of the connectivity-description (see subclause 5.2.3.1.1) be defined as an absolute reference to the grid, i.e. an absolute position (*u*,*v*,*w*) and discrete border direction. As shown in Table 16 (c), the discrete border direction requires 3 bits to be encoded. Each of the indices (*u*,*v*,*w*) defining the position of V_S can be encoded on *n*-bits, where *n* (n_u,n_v,n_w) depends on the largest ROI size which is $szDefROI_d + 1$, as explained in B.1.2:

$$n_d = \lfloor \log_2(szDefROI_d + 1) \rfloor + 1, \quad d \in \{u, v, w\} \tag{3}$$

5.2.3.1.1 Decoding the Connectivity bits

The decoding of a multi-resolution connectivity-wireframe consists basically in decoding sequentially each single-resolution connectivity-wireframe apart. A single-resolution connectivity-wireframe is encoded on ROI basis, be it a single ROI or several. The connectivity-wireframe from each ROI can be decoded in random order in view-dependent decoding scenarios. The coded connectivity-description from each ROI may consist of one or several patches. Each patch contains (1) a starting vertex identified by its discrete position (*u*,*v*,*w*) and discrete orientation (border direction), (2) followed by a sequence of relative connectivity symbols computed between an ancestor and its outgoing siblings. The traversal from the ancestor to the siblings is based on a first-in-first-out (FIFO) approach, as illustrated in Figure 35. The patch coding/decoding stops automatically when all the outgoing connectivities of the sibling vertices have been consumed inside the ROI. In order to avoid coding twice the connectivity of the vertices at the border in between two adjacent ROIs, the

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

domain of the ROI is [*min*, *max*) for each of the {*u*,*v*,*w*} directions. For the last ROI in any direction $d \in \{u,v,w\}$, the domain of the ROI is [min, max] in direction d.

For each resolution level *L* the ROIs are coded in ascending order as given by the following formula:

$$i + j \times nROIs_U^L + k \times (nROIs_U^L + nROIs_V^L), \qquad (4)$$

with $i \in [0, nROIs_U^L - 1]$, $j \in [0, nROIs_V^L - 1]$, $k \in [0, nROIs_W^L - 1]$. First, *i* is incremented, followed by *j*, and next by *k*.

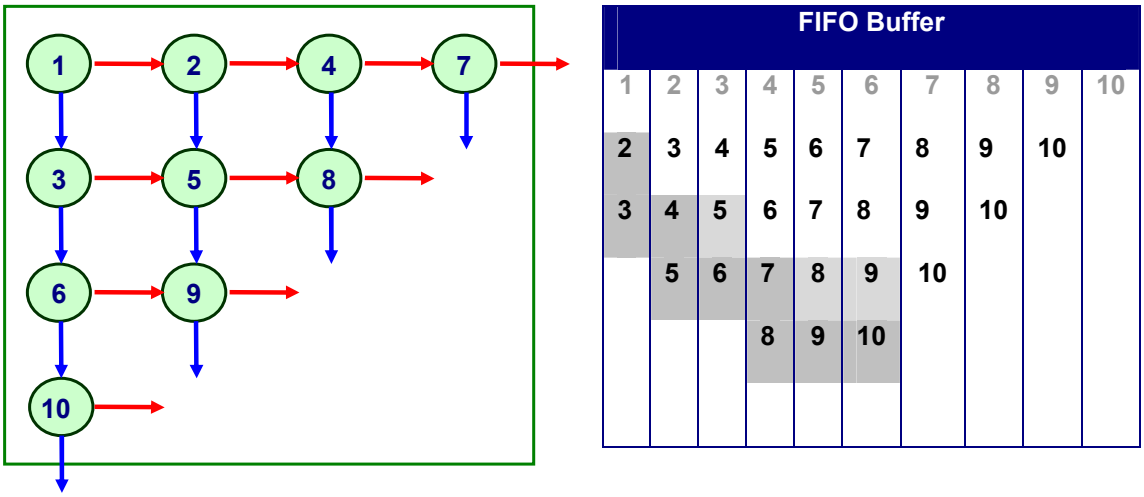


Figure 35 — Coding the LSD with a FIFO buffer

The decoding scenario for a patch illustrated by the example from Figure 35 is as follows: Retrieve the discrete position and discrete direction of the 1st vertex (starting vertex V_S) and store it in a *FIFO*. Consume the first vertex from the *FIFO*, retrieve the connectivity symbol leading to vertex 2 (reached via the outgoing link -L_{N1}-) and the connectivity symbol leading to vertex 3 (reached via outgoing link -L_{N2}-), compute the discrete positions of the vertices 2 and 3, and store both vertices 2 and 3 in the *FIFO*. Next pop vertex 2 from the *FIFO*, retrieve the two connectivity symbols leading to vertices 4 and 5, compute the discrete positions of the vertices 4 and 5, and insert them in the *FIFO*. In case the retrieved symbol leads to an outgoing connectivity vector pointing to a vertex V_O that was already visited before, or to a vertex V_O that lies outside the *ROI*, the visited vertex V_O will not be put in the *FIFO*. When consuming vertex 3 from the *FIFO*, retrieve the next two connectivity symbols leading to vertex 5 and 6, but insert in the *FIFO* only vertex 6, since vertex 5 has already been visited. The decoding scenario of a patch will stop when all vertices in the *FIFO* are consumed. Due to this implicit stopping criterion, only one byte-counter specifying the total length of all the coded patches belonging to the ROI is sufficient. For the other patches of the ROI, the same scenario is repeated.

5.2.3.1.2 Deriving the discrete positions of the vertices

In Table 17, the column header of the table has the labels of the border directions related to a vertex playing the role of an ancestor, while the row header displays the same border directions, but related to a sibling vertex. The table gives all the possible combinations between the directions of the ancestor vertex and the sibling vertices. Combinations that are not possible have been shown with an exclamation mark. For any direction of the ancestor vertex, there are always only three valid border directions (with different combinations) for the sibling vertices.

For each valid case, the table gives also the position (*u*,*v*,*w*) in the reference-grid of the sibling vertex relative to the ancestor's vertex (only the position coordinate in the reference-grid of the sibling that differs from the ancestor's position is shown), and the connectivity vector from the sibling back to the ancestor.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2006 – All rights reserved
Not for Resale

Table 17 — Relation between the discrete border directions of two connected vertices and their discrete coordinates (*u,v,w*). Notice the change of indices of the connectivity vectors between adjacent vertices such that the normal vector computed with Equation 15 points outwards.

	Back		Front		Left		Right		Down		Up	
	$-L_{N1}-$	$-L_{N2}-$	$-L_{N1}-$	$-L_{N2}-$	$-L_{N1}-$	$-L_{N2}-$	$-L_{N1}-$	$-L_{N2}-$	$-L_{N1}-$	$-L_{N2}-$	$-L_{N1}-$	$-L_{N2}-$
Back	*v -1* $-L_{P1}-$	*w +1* $-L_{P2}-$	!	!	!	*v -1* *u +1* $-L_{P1}-$	same $-L_{P1}-$	!	same $-L_{P2}-$	!	!	*w +1* *u +1* $-L_{P2}-$
Front	!	!	*w -1* $-L_{P1}-$	*v +1* $-L_{P2}-$	!	same $-L_{P2}-$	*v +1* *u -1* $-L_{P2}-$	!	*w -1* *u -1* $-L_{P1}-$	!	!	same $-L_{P1}-$
Left	same $-L_{P2}-$	!	!	*u +1* *v +1* $-L_{P2}-$	*w -1* $-L_{P1}-$	*u +1* $-L_{P2}-$	!	!	!	*w -1* *v +1* $-L_{P1}-$	same $-L_{P1}-$	!
Right	*u -1* *v -1* $-L_{P1}-$	!	!	same $-L_{P1}-$	!	!	*u -1* $-L_{P1}-$	*w +1* $-L_{P2}-$	!	same $-L_{P2}-$	*w +1* *v -1* $-L_{P2}-$	!
Down	!	*u -1* *w +1* $-L_{P1}-$	same $-L_{P1}-$	!	same $-L_{P2}-$	!	!	*v +1* *w +1* $-L_{P2}-$	*u -1* $-L_{P1}-$	*v +1* $-L_{P2}-$	!	!
Up	!	same $-L_{P2}-$	*u +1* *w -1* $-L_{P2}-$	!	*v -1* *w -1* $-L_{P1}-$	!	!	same $-L_{P1}-$	!	!	*v -1* $-L_{P1}-$	*u +1* $-L_{P2}-$

5.2.3.1.3 The Cyclic Mode

The choice of the reference-grid as described in subclause 4.3.4.1.1 depends on the type of mesh that needs to be represented; it can range from uniform to highly irregular distributions of the grid points. In addition, the topology of the reference-grid can be defined as open, closed or folded. The closed reference-grid is designed by setting the grid points of the last reference-grid plane to the same coordinates as the corresponding grid points of the first reference-grid plane, in the "*U*" direction, or in two directions ("*U*" and "*V*") simultaneously. The folded reference-grid is can be cyclic in the "*U*" direction and connected in the "*V*" direction, i.e. half of the grid points of the first (respectively last) reference-grid line are connected to the corresponding points of the other half by folding the line in the middle.

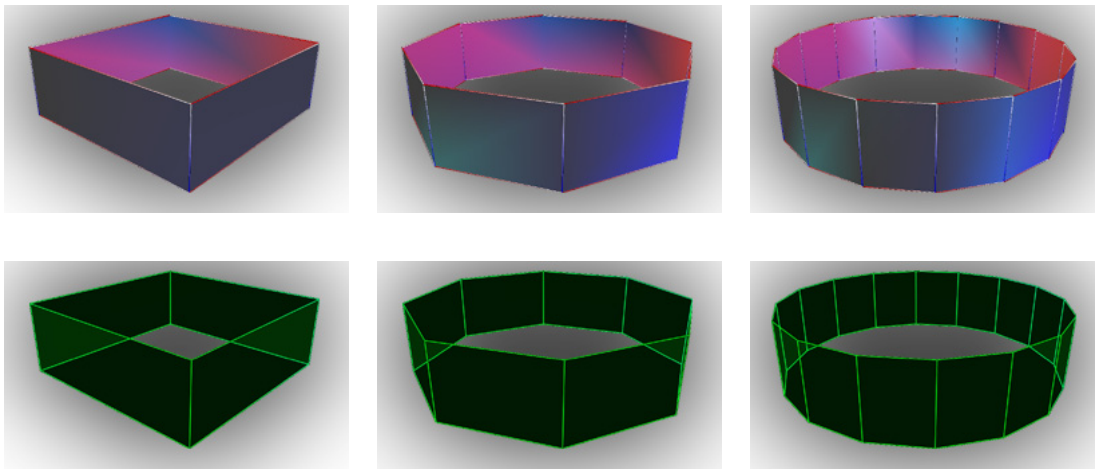
A connectivity-wireframe is cyclic when the connectivity vectors between the vertices span from the vertices located on the last (respectively first) reference-grid planes to those located on the first (respectively last) reference-grid planes. A connectivity-wireframe is folded when it is cyclic in the "*U*" direction and connected in the "*V*" direction, i.e. the connectivity-vectors from the vertices located on the first (respectively last) folded reference-grid line span to corresponding vertices on the folded line.

The cyclic/folded mode of a mesh is specified by the *cyclic_folded* variable from the *MeshGridDecoderConfig* class. As illustrated in Table 12, there is one non-cyclic mode, 4 cyclic/folded modes. A cyclic, respectively folded, mesh should be attached to a closed, respectively folded, reference-grid. When the mesh is cyclic, the connectivity-wireframe decoder will position each vertex that would normally be attached to the last reference-grid plane on the first reference-grid plane, in order to avoid the duplication of these vertices. In addition, when a mesh is folded, one vertex from each pair of duplicated vertices, i.e. pairs of vertices located on the folded reference-grid lines, is removed.

An example of a multi-resolution model cyclic in a single direction, in two directions and folded is shown in Figure 36, Figure 37 and Figure 38 respectively.

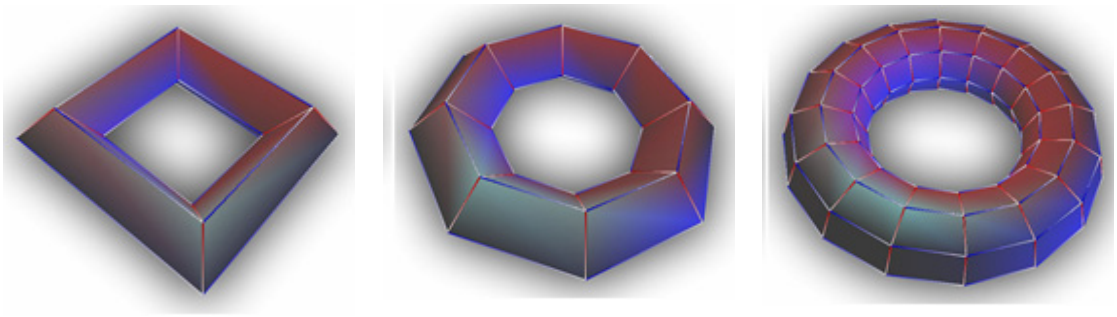
Note however, that even when the *cyclic_folde* mode flag is set to non-cyclic, it is still possible to decode a cyclic or folded connectivity-wireframe defined on top of either an open, closed or folded reference-grid, but in this situation the decoder will not take care of overlapping vertices.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



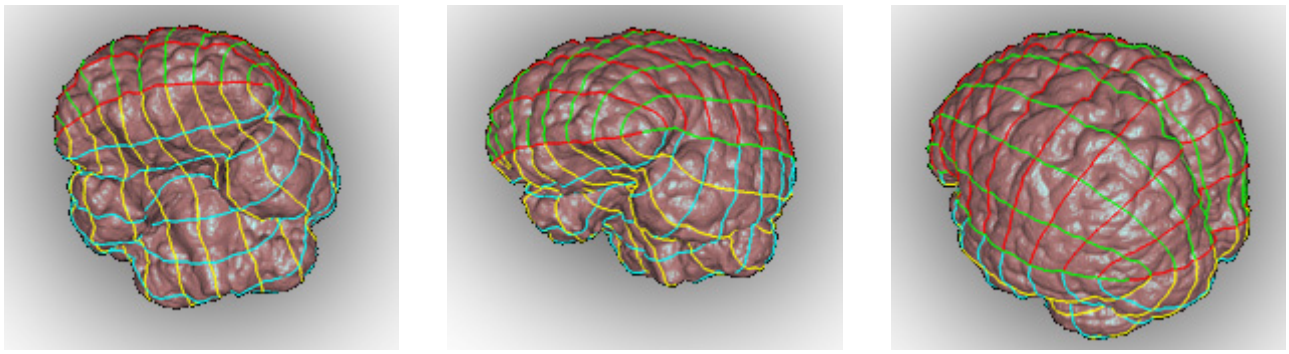
(a) (b) (c)

Figure 36 — Example of a multi-resolution mesh cyclic in one direction, and its corresponding reference-grid.



(a) (b) (c)

Figure 37 — Example of a multi-resolution mesh cyclic in two directions.



(a) (b) (c)

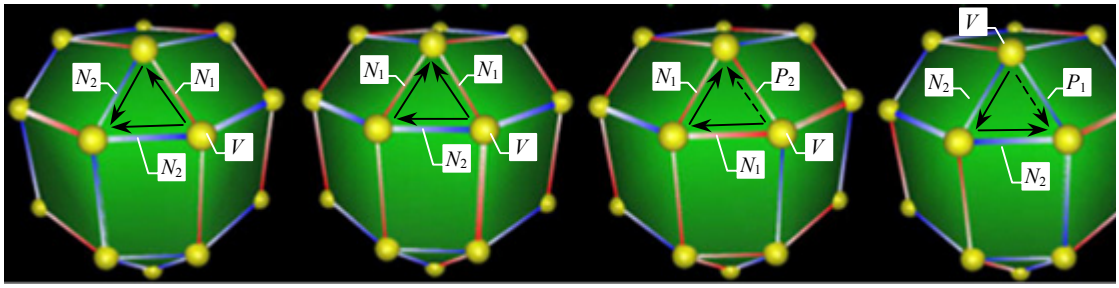
Figure 38 — Different views of a folded mesh.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2006 – All rights reserved

5.2.3.1.4 Rules for deriving the surface primitives from the connectivity-wireframe

The polygonal representation of a MeshGrid model consists in triangulating the connectivity-wireframe, which can be seen as a union of connectivity circuits. A connectivity circuit corresponds to the shortest path formed by navigating from a starting vertex v_S to its neighbors following the connectivity vectors and back to vertex v_S.

In order to unambiguously identify the shape and the orientation of a connectivity circuit *CC*, such that the right surface primitive can be inserted, a set of connectivity rules should be designed. There are five types of primitives: triangles, quadrilaterals, pentagons, hexagons and heptagons, which can be identified by means of connectivity rules. As an example, a set of connectivity rules is given for the triangle primitive, Figure 39 showing the graphical cases, while Table 18 giving the connectivity conditions. The connectivity vectors drawn in Figure 39 use the same color convention as in Figure 34 and in Table 17. A complete set of rules for all five types of primitives is described in [71].



(a) (b) (c) (d)

Figure 39 — Images for the connectivity cases corresponding to the triangle primitive.

Table 18 — Connectivity cases for the triangle primitive

Case	Rule
(a)	$N_1N_2 = N_2$
(b)	$N_2\ N_1 = N_1$
(c)	$N_1N_1 = P_2$ or $N_1N_1N_1 = V$
(d)	$N_2N_2 = P_1$ or $N_2N_2N_2 = V$

5.2.3.2 Reference-Grid Decoder

The general representation of the Reference-Grid description is a vector field $(x(u,v,w),y(u,v,w),z(u,v,w))$ defined on the regular discrete 3D space (u,v,w). Each component is coded separately, by means of a progressive multi-resolution algorithm based on a combination of a 3D wavelet transform and an intra-band wavelet coder, called Cube Splitting, which is the 3D extension of the SQP (Square Partitioning) algorithm described in [59]. This coding/decoding approach supports quality, resolution scalability, and ROI coding/decoding [2], [58], [59].

Subclause 5.2.3.2.1 describes the particular type of filters and down-sampling/up-sampling operations used for the wavelet decomposition and reconstruction, while the Cube Splitting algorithm is described in subclause 5.2.3.2.2.

5.2.3.2.1 The 3D Wavelet Decomposition

The same 3D-wavelet decomposition is applied independently to each of the $x(u,v,w)$, $y(u,v,w)$, $z(u,v,w)$ coordinates of the 3D reference grid.

© ISO/IEC 2006 – All rights reserved

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

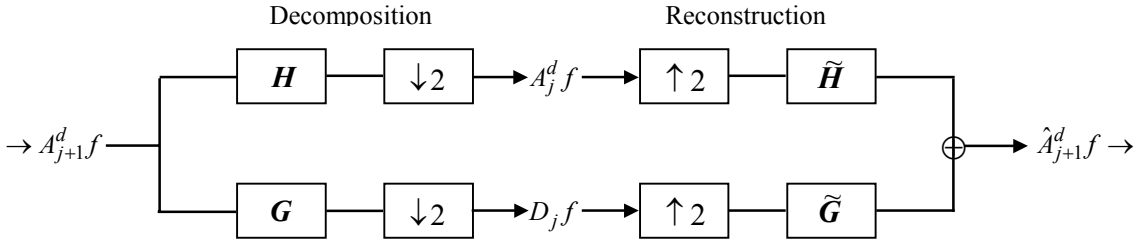


Figure 40 — Wavelet decomposition and reconstruction of a 1D signal.

For the 3D wavelet decomposition, the same analysis and synthesis 1D filters are used for each of the *u*, *v*, *w* directions. Two wavelet filters are supported, and the choice of the filter is specified by the *filterType* flag explained in section 5.2.2.2. Conform to the block scheme [54] shown in Figure 40, the two analysis low-pass and band-pass wavelet filters are respectively:

$$H(n) = \{1 \mid n = 0\},\ G(n) = \{-0.5, 1, -0.5 \mid n = -1, 0, 1\},\ \text{for } \textit{filterType} = 0,$$

Equation 16

$$H(n) = \{1 \mid n = 0\},\ G(n) = \left\{\tfrac{1}{16}, 0, -\tfrac{9}{16}, 1, -\tfrac{9}{16}, 0, \tfrac{1}{16} \mid n = -3, -2, -1, 0, 1, 2, 3\right\},\ \text{for } \textit{filterType} = 1,$$

Equation 17

The synthesis low-pass and band-pass wavelet filters are:

$$\tilde{H}(n) = \{0.5, 1, 0.5 \mid n = -1, 0, 1\},\ \tilde{G}(n) = \{1 \mid n = 0\},\ \text{for } \textit{filterType} = 0,$$

Equation 18

$$\tilde{H}(n) = \left\{-\tfrac{1}{16}, 0, \tfrac{9}{16}, 1, \tfrac{9}{16}, 0, -\tfrac{1}{16} \mid n = -3, -2, -1, 0, 1, 2, 3\right\},\ \tilde{G}(n) = \{1 \mid n = 0\}\ \textbf{, for } \textbf{\textit{filterType}} \textbf{ = 1.}$$

Equation 19

The wavelet filters are the same as the filters used for the hierarchical interpolation of the reference-grid points explained in subclause 4.3.4.3.1.

Further, in Figure 40, $A_j^d f(n)$ and $D_j f(n)$ are the discrete approximation and the detail signals respectively at the resolution $2^j, -L \le j \le -1$ of the input signal *f*, where *L* is the number of decomposition levels, and $A_0^d f(n) = f(n)$.

The low-pass filter *H* and the band-pass filter *G* are applied on the even and odd samples respectively of $A_{j+1}^d f(n)$ and the synthesis filters $\tilde{H}$ and $\tilde{G}$ are applied on the even and odd samples respectively of the low-pass and band-pass components $A_j^d f(n)$ and $D_j f(n)$.

The grid can be decoded at any resolution *j* if and only if the corners of the grid are stored in any $A_j^d f, -L \le j < 0$. There are some situations in which this constraint is not satisfied with the classical implementation of the pyramidal algorithm [54] discussed above. The wavelet analysis/synthesis filters given by Equation 17 and Equation 19 shall only be used when the constraint is satisfied for all decomposition levels $A_j^d f(n)$: the length of the discrete approximation signal $A_j^d f$ is odd. To solve this problem, a customized implementation of the pyramidal algorithm involves, in some situations, non-uniform down-sampling and up-sampling operations (see Figure 41(b,c)), coupled with analysis/synthesis filters that are different than the $H, G, \tilde{H}, \tilde{G}$ given by Equation 16 and Equation 18.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

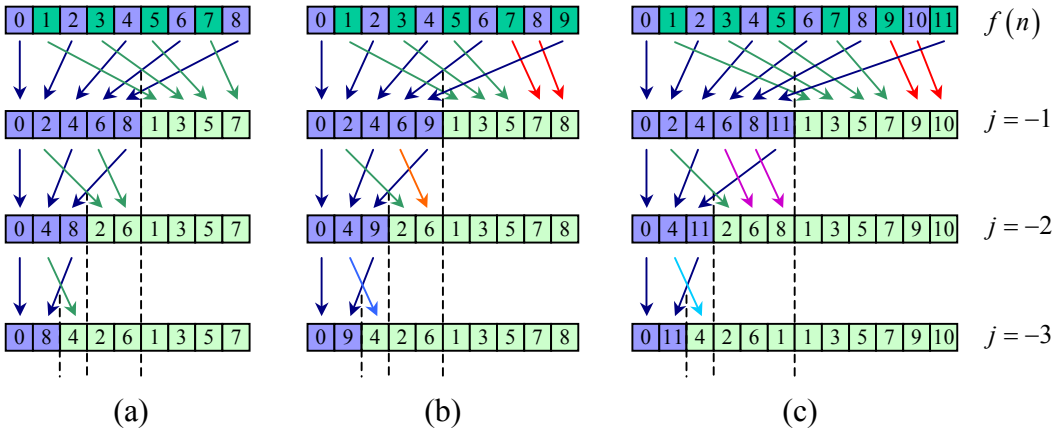


Figure 41 — Graphical illustration of the wavelet decomposition for odd and even-length signals.

As long as the length of the discrete approximation signal $A_j^d f$ is odd (see Figure 41(a)), there is no need to perform a non-uniform down-sampling operation, and the classical pyramidal algorithm can be used. If at some resolution level $r, -L < r \le 0$ the length of $A_r^d f$ is even, then apart from the common situation in which G given by Equation 18 is used, the analysis band-pass filters $G^1_{|r|+1}, G^2_{|r|+1}$ have to be used as well, and non-uniform down-sampling has to be applied for the last samples. This operation has to be repeated for all the resolution levels $p, -L < p \le r$. The filter coefficients of $G^1_{|p|+1}, G^2_{|p|+1}$ depend on the length of the discrete approximation signal $A_p^d f$ (whether it is an odd or an even number) and on the resolution level p . The additional filters used to derive the detail $D_{p-1} f$ starting from the discrete approximation $A_p^d f$ are:

$$G_p^1(n) = \{-c_2, 0, 1, -c_1 \mid n = -2, -1, 0, 1\},\ G_p^2(n) = \{-c_4, 1, 0, -c_3 \mid n = -1, 0, 1, 2\} \quad (16)$$

if the length of $A_p^d f$ is even, and:

$$G_p^1(n) = \{-c_6, 1, -c_5 \mid n = -1, 0, 1\} \quad (17)$$

if the length of $A_p^d f$ is odd. The constants $c_1, ..., c_6$ verify the relations:

$$c_1 = 1 - c_2,\ c_3 = 1 - c_4,\ c_4 = 2c_2, \text{ and } c_5 = 1 - c_6 \quad (18)$$

The constants c_2, c_6 are given by $c_2 = c_6 = x(n)/y(n)$, where $x(n)$ satisfies the recurrence $x(n) = x(n-1) + 2^{n-1}$, and $y(n)$ satisfies the recurrence $y(n) = y(n-1) + 2^{n-1}$ if the length of $A_p^d f$ is even, respectively $y(n) = y(n-1) + 2^n$ if the length of $A_p^d f$ is odd, with $x(0) = 1, y(0) = 3$ and $n = r - p$.

© ISO/IEC 2006 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.3.2.2 Intra-band wavelet coding algorithm

The coefficients generated by the wavelet transform undergo a scaling (see subclause 5.2.2.25) before they are coded with the Cube Splitting algorithm. The cube splitting algorithm is the straightforward extension of the SQP coding algorithm [59] to 3D. The coding of the coefficients is done bit-plane by bit-plane in a depth-first strategy. With each bit-plane (b=0 is the lowest bit-plane), a threshold T_b is associated with $T_b=2^b$. Conceptually, there are two coding passes for each bit-plane (except for the first bit-plane): a significance pass and a refinement pass. A cube C is said to be significant for a certain threshold T if:

$$\underset{c(u,v,w)\in C}{MAX}\left(c(u,v,w)\geq T\right) \tag{5}$$

When a cube C is found non-significant, the NSG symbol is written into the bitstream. Otherwise, the SGN symbol is written and the cube is further split into 8 sub-cubes, visited in terms of the (u,v,w) reference axes in the following o rder: (0,0,0), (1,0,0), (0,1,0), (1,1,0), (0,0,1), (1,0,1), (0,1,1), (1,1,1).

During the splitting, each dimension is divided into two intervals. Let A be the smallest coordinate value, respectively B the highest coordinate value for a certain dimension {u,v,w} of the cube. The corresponding size S in that dimension is equal to $S = B - A + 1$. The min values (A_{Left}, A_{Right}), max values (B_{Left}, B_{Right}), and the sizes (S_{Left}, S_{Right}) of the intervals can be computed as:

$$\begin{aligned} &S_{Left} = \lfloor (S+1)/2 \rfloor, \quad S_{Right} = \lfloor S/2 \rfloor, \\ &A_{Left} = A, \quad B_{Left} = A_{Left} + S_{Left} - 1, \\ &A_{Right} = A + S_{Left}, \quad B_{Right} = A_{Right} + S_{Rigth} - 1. \end{aligned} \tag{6}$$

If a cube has been found significant at a previous bit-plane, no SGN symbol is written in the stream. When the algorithm reaches the voxel level (i.e. no further splitting is possible), it first checks whether the voxel was already significant for higher bit-planes. If this is the case, the refinement symbol (BIT1 or BIT0) will be written into the stream. Otherwise, the algorithm will determine whether the voxel is significant for the current threshold value. If this is the case, the significant symbol SGN is written and is followed by the symbol for the sign (+,-) of the voxel-value. If both the previous cases are not valid, the non-significant symbol NSG will be written.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

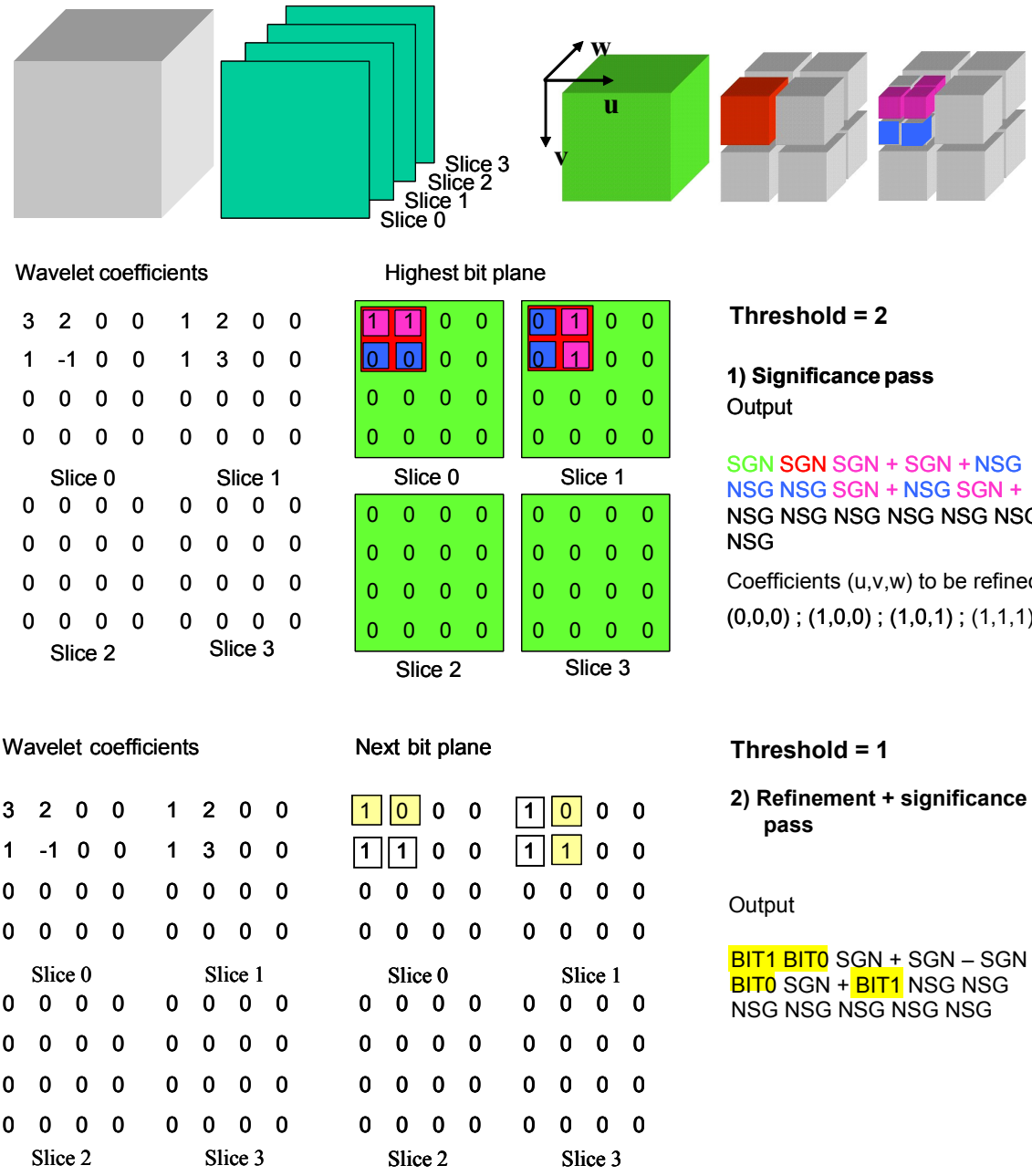


Figure 42 — Graphical representation of the significance and refinement coding passes.

The convention used in the MeshGrid stream for coding the symbols is shown in the following table:

Symbol	Bit value
SGN	1
NSG	0
BIT1	1
BIT0	0
+	1
-	0

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.3.3 Vertices-Refinement Decoder

The vertices are located at the intersection positions between the grid-lines and the object contour, at a certain ratio (the *offset*) in between two reference-grid points (see subclause 4.3.4.3.2). The *offset* (see Figure 23) has a default value of 0.5, but can be modified by the refinement description to fit a known position. The ratio can vary between [0,1).

The refinement description consists of two parts: (1) the inter-resolution refinement bit (*IRR*), and (2) the *offset*.

The significance of the *IRR* is shown in **Figure** 43. When decoding the next higher resolution level, some vertices may change their current discrete grid position (*u*,*v*,*w*), and migrate towards a neighboring grid position belonging to the higher resolution level. As a result of the migration, one of the {*u*,*v*,*w*} indices will change. The migration occurs along the grid line in the direction of the border. As illustrated in Figure **43**, only the vertices with bit values larger than 0 will migrate.

For a specified resolution level (*l*) the default number of *IPR* bits for each vertex is one. For any vertex *v*(*u*,*v*,*w*), in the particular case that (1) the number of slices ($nSlices_d$) of any direction $d \in \{u,v,w\}$ of the immediate higher level (*l*+1) is a even number, (2) the grid position of vertex *v* in direction *d* is equal to $(nSlices_d - 2)$, and (3) the discrete border direction of vertex *v* is oriented towards the positive axis of direction *d,* then there will be two *IRR* bits necessary to encode the repositioning value of vertex *v*, since two additional slices will be inserted between the slice at indices $(nSlices_d - 2)$ and $(nSlices_d - 1)$. This particular case has been explained in subclause 5.2.3.2.1, with a graphical example in Figure 41.

The *IRR* bits are necessary for all resolution levels, except for the last resolution level. A 2-bit flag (*repositionBits* explained in subclause 5.2.2.18) is present at each resolution level (except last) specifying whether default values are set for the *IRR* bits. If default values are specified, no *IRR* bits are encoded for that resolution level. Otherwise, the *IRR* bits are present in the stream. A default value of 0 or 1, indicates that all the *IRR* bits are 0 or 1, respectively.

For any ROI, the coding order in the stream of the *IRR* bits is the same as the order of the decoded vertices (see subclause 5.2.3.1.1) of the ROI.

If the number of quantization bits for the *offset* (*nBits* explained in subclause 5.2.2.23) is larger than 0, then the *offset* bits are present in the stream. Otherwise they are not available. If the full refine flag is set in the bitstream (*bFull* explained in subclause 5.2.2.23), then the *offset* bits are coded for each resolution level. Otherwise, the *offset* bits will only be coded for the last resolution level. The *offset* bits are stored bit-plane by bit-plane in the same order as the decoded vertices (see subclause 5.2.3.1.1).

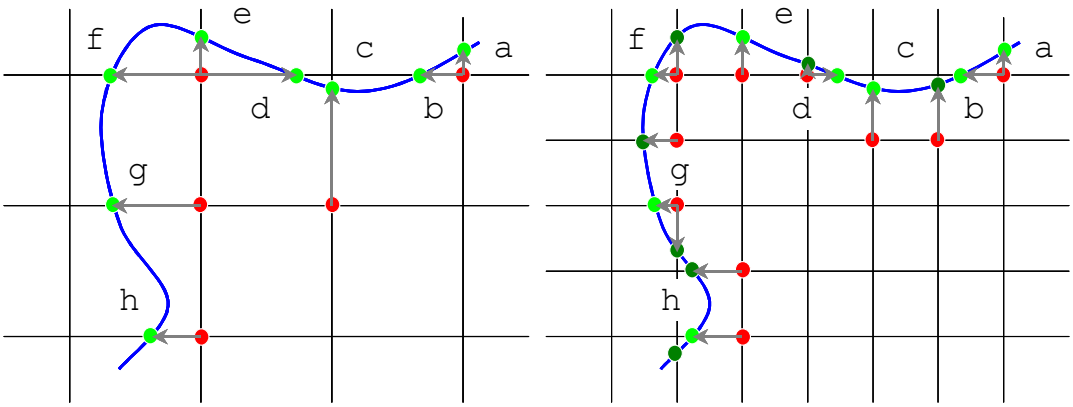


Figure 43 — Demonstration of vertex migration via the Inter-Resolution Refinement bits. The labelled vertices from the lower resolution Mesh-Description (left image) will migrate (some of them) to new reference-grid positions in the next higher resolution level (right image). For the sequence of vertices abcdefgh, the refinement bits are: 00110110, which means that vertices a, b, e, and h will not migrate while the others do.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.2.3.4 ROI decoding for view-dependent functionality

As described in the beginning of subclause 5.1, the MeshGrid models are encoded at each resolution level in separate *region of interests* (ROIs), such that random decoding of the ROIs is possible to allow view-dependent scenarios.

The mesh description is encoded in the “spatial-domain” representation of the MeshGrid model, while the grid description is encoded in the “wavelet-domain” representation of the MeshGrid model. In a view-dependent scenario some parts of the model, which are identified in the “spatial-domain”, have to be decoded with a higher priority than the others. Therefore, a correspondence has to be determined between the “spatial-domain” ROIs (SROIs) and the “wavelet-domain” ROIs (WROIs) such that the appropriate grid description is decoded for the mesh description inside the SROIs. The correlation between a SROI defined at a certain resolution level and its WROIs is illustrated in Figure 44.

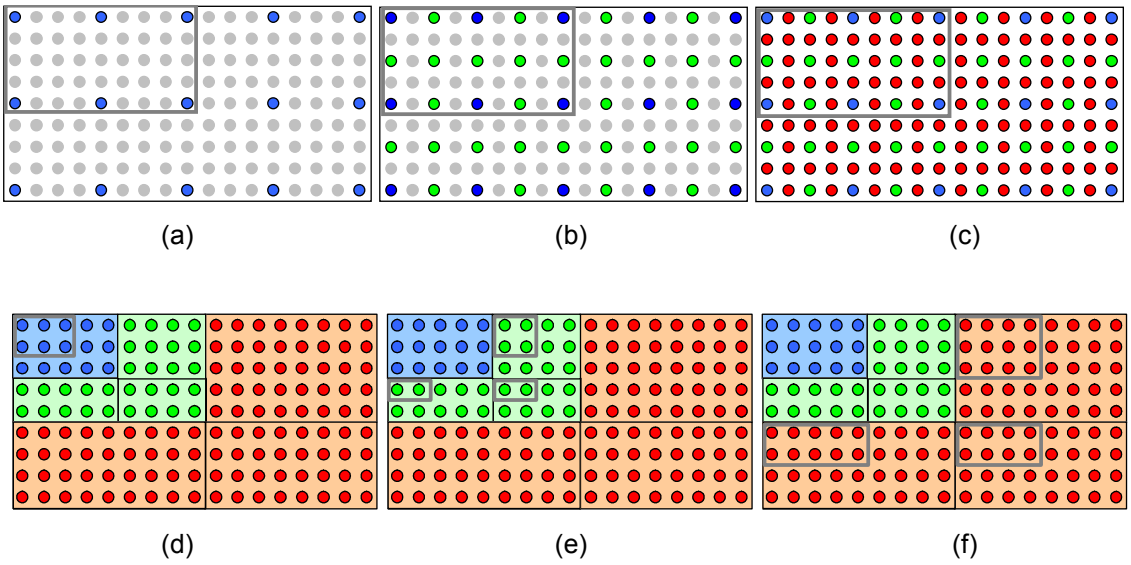


Figure 44 — A slice through the 3D reference-grid illustrating: (1) the distribution of the grid-points in the spatial domain (a, b, c) (resolution l, l+1, l+2) and in the wavelet domain (d, e, f), (2) the correspondence between a ROI defined at different resolutions in the spatial domain and its corresponding WROIs in the wavelet domain.

For each resolution level of the wavelet-decomposed reference-grid, the WROIs that are related to the same spatial domain SROI are grouped together and encoded as on tile. Therefore, for any resolution level *L* and direction $d \in \{U,V,W\}$ the number of tiles ($nTiles_d^L$) are equal to the number of ROIs ($nROIs_d^L$). Corresponding SROIs and tiles yield the same index $idx_d^L \in [0, nROIs_d^L - 1]$ when encoded in the bitstream. The tiles are coded with the Cube Splitting algorithm, described in subclause 5.2.3.2.2, in the same ascending order as the SROIs using the following formula:

$$i + j \times nROIs_U^L + k \times \left(nROIs_U^L \times nROIs_V^L\right), \tag{7}$$

with $i \in [0, nROIs_U^L - 1]$, $j \in [0, nROIs_V^L - 1]$, $k \in [0, nROIs_W^L - 1]$. First, i is incremented, followed by j, and next by k.

The number of ROIs is stored in the *nROIs* field of the MeshGridDecoderConfig class (see subclause 5.2.2.22) and the approach to compute the number of ROIs for the lower resolution levels and to uniformly distribute these ROIs is explained in subclause B.1.2.

© ISO/IEC 2006 – All rights reserved

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

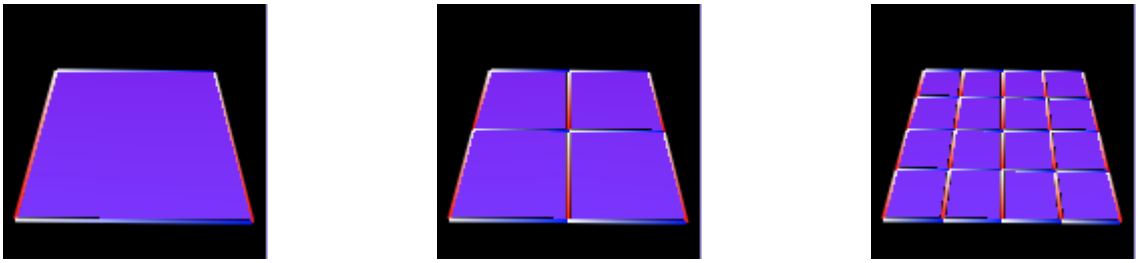
5.2.3.5 Decoding Quadrilateral MeshGrid

According to the *sameBorderOrientation* flag from the *MeshGridDecoderConfig* class (subclause 5.2.2.2) quadrilateral meshes can be classified into: (1) type1 (generic) – *sameBorderOrientation* equals '0' – when the vertices may have different discrete border directions (see Figure 34), and (2) type2 – *sameBorderOrientation* equals '1' – when all the vertices have the same discrete border direction.

For the type2 quadrilateral MeshGrid the *uniformSplit* flag from the *MeshGridDecoderConfig* class can have the value '1', which means that any higher resolution level quadrilateral mesh can be obtained from a lower resolution level mesh by uniformly splitting each quad recursively into four sub-quads as illustrated in Figure 45. In addition if the *hasConnectivityInfo* flag from the *MeshGridDecoderConfig* class has the value '0' – meaning that there is no connectivity-wireframe stored in the stream – then a default type2 quadrilateral connectivity-wireframe shall be constructed for the first resolution level. In this case the *uniformSplit* and *sameBorderOrientation* flags are set to '1'.

The reference-grid corresponding to the type2 quadrilateral MeshGrid with *hasConnectivityInfo* flag set '0' has the number of slices – *nSlices* from the *MeshGridDecoderConfig* class – in one of the directions {*u*,*v*,*w*} either equal to '2' – a double layer reference grid –, or equal to '1' – a single layer reference grid. For a double layer reference-grid the default type2 quadrilateral connectivity-wireframe shall be obtained by attaching a vertex to each reference-grid point belonging to the first layer of points i.e. the layer with the property that all the contained reference-grid points have one of the indices {*u*, *v*, *w*} equal to '0', and positioning these vertices on the reference-grid lines connecting the corresponding grid points from the first layer to the second layer. When the reference-grid is single layer, the default type2 quadrilateral connectivity-wireframe shall be obtained by attaching a vertex to each reference-grid point. The coordinates of the vertices can be computed as explained in subclause 4.3.4.3.2.

When the *hasRefineInfo* flag from the *MeshGridDecoderConfig* class (subclause 5.2.2.2) is set to '1' and there is a type2 quadrilateral MeshGrid with *hasConnectivityInfo* flag set '0', then the offsets (subclause 5.2.3.3) are specified only for the first resolution level of the mesh. The order of the offsets in the stream is the same as the indexing order of the reference-grid positions within the layer where the vertices are attached (the index corresponding to the layer is kept constant). The reference-grid points are indexed by incrementing two of the {*u*, *v*, *w*} indices in the order: *U* direction first, *V* direction second, and *W* direction third. The offset of each vertex corresponding to any higher resolution levels is computed as the average value of the offsets corresponding to the neighbour vertices' belonging to a lower resolution level. Namely, given V_1, V_2, V_3 and V_4 four vertices belonging to resolution level *l* (see Figure 45), the offset of a vertex *V* belonging to level *l* + 1 located on a grid line of level *l* is computed as the average value of the offsets of V_1 and V_2, respectively the offset of vertex *V*' belonging to level *l* + 1 located on a grid line of level *l* + 1 is computed as the average value of the offsets of V_1, V_2, V_3 and V_4.



(a) (b) (c)

Figure 45 — Example of a multi-resolution mesh with the connectivity-wireframe obtained by uniformly splitting each quad recursively into four sub-quads.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

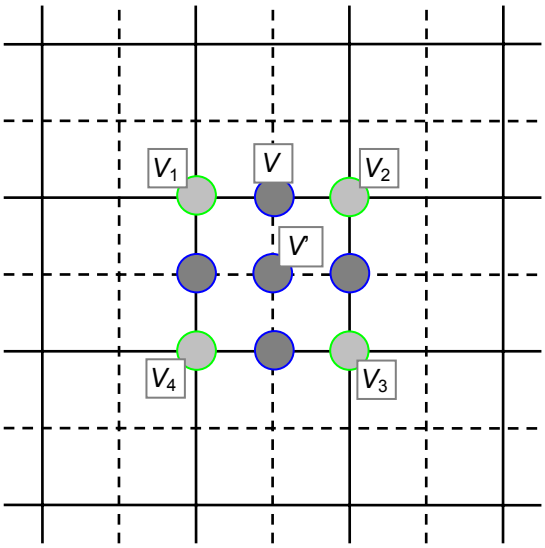


Figure 46 — Computation of the offsets for the type2 quadrilateral MeshGrid.

5.3 Depth Image-Based Representation

5.3.1 Octree Compression

5.3.1.1 Overview

The OctreeImage node in Depth Image-Based Representation defines the octree structure and their projected textures. Each texture, stored in the images array, is defined through DepthImage node with SimpleTexture. The other fields of the OctreeImage node can be compressed by octree compression.

5.3.1.2 Octree

5.3.1.2.1 Syntax

```
class Octree ()
{
   OctreeHeader ();
   aligned bit (32)*  next;
   while (next == 0x000001C8)
   {
      aligned bit (32) octree_frame_start_code;
      OctreeFrame(octreeLevel);
      aligned bit (32)*  next;
   }
}
```

5.3.1.2.2 Semantics

The compressed stream of the octree contains an octree header and one or more octree frame, each preceded by **octree_frame_start_code**. The value of the **octree_frame_start_code** is always 0x000001C8. This value is detected by look-ahead parsing (next) of the stream.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.3.1.3 OctreeHeader

5.3.1.3.1 Syntax

```
class OctreeHeader ()
{
   unsigned int (5) octreeResolutionBits;
   unsigned int (octreeResolutionBits) octreeResolution;
   int octreeLevel = ceil(log(octreeResolution)/log(2));

   unsigned int (3) imageNumBits;
   unsigned int (imageNumBits) numOfImages;
}
```

5.3.1.3.2 Semantics

This class reads the header information for the octree compression.

The **octreeResolution**, whose length is defined by **octreeResolutionBits**, contains the value of octreeResolution field of OctreeImage node. This value is used to derive the octree level.

The **numOfImages**, which is **imageNumBits** long, describes the number of images used in the OctreeImage node. This value is used for the arithmetic coding of image index for each voxel of the octree. If the value of **imageNumBits** is 0, then the image index symbols are not coded in OctreeFrame().

5.3.1.4 OctreeFrame

5.3.1.4.1 Syntax

```
class OctreeFrame (int octreeLevel)
{
   if(imageNumBits==0)  parentImage=255;
   else                 parentImage=0;
   for (int curLevel=0; curLevel < octreeLevel; curLevel++)
   {
      for (int voxelIndex=0; voxelIndex < nVoxelsInCurLevel; voxelIndex++)
      {
         int voxelSym = ArithmeticDecodeSymbol (contextID);
         if (parentImage == 0)
         {
            curImage = ArithmeticDecodeSymbol (imageContextID);
         }
      }
   }
   for (int voxelIndex=0; voxelIndex < nVoxelsInCurLevel; voxelIndex++)
      if (parentImage == 0)
         curImage = ArithmeticDecodeSymbol (imageContextID);
}
```

5.3.1.4.2 Semantics

This class reads a single frame of octree in a breadth first traversal order. Starting from the 1st node from the level 0, after reading every voxels in the current level, the number of voxels (nVoxelsInCurLevel) in the next level is known by counting all the 1's in each voxel symbol. In the next level, that number of voxels will be read from the stream.

For decoding of each voxel, an appropriate contextID is given, as described in subclause 5.3.1.6.

The parentImage and curImage are internal variables within each voxel in each level. The curImage is the image index for the current voxel in the current level. The parentImage is the image index of the parent, in the previous level, of the current voxel (the curImage of the parent voxel). When the bitstream does not contain

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

the voxel image index information, the imageNumBits will be 0. In this case, the parentImage for every voxel shall be initialized to 255. Otherwise, the initial value of the parentImage for every voxel shall be 0.

If the image index for the parent voxel (parentImage) is not defined, then the image index for the current voxel (curImage) is also read from the stream, using the context for image index, defined by imageContextID. If a non-zero value is retrieved (the image index is defined), then this value will also be copied to every image index (curImage) of its children voxels in the following levels. In case of the root voxel, where curLevel is 0 and voxelIndex is 0, the value of parentImage depends on the value of **imageNumBits** in OctreeFrame(). If the imageNumBits is 0, then the parentImage of every voxel is a non-zero dummy value. This is the case where the voxel image indices are not used. If the imageNumBits has non-zero value, then the parentImage of every voxel is by default 0.

After decoding every voxel, the image index will be assigned to every leaf node of the octree that has no image index value assigned by the parent voxel yet. (zeros in leaf nodes can be interpreted by renderer in an arbitrary way).

The order of the reference images (streams) is essential: image indices for voxels assume the streams are numbered in a fixed order.

5.3.1.5 Adaptive Arithmetic Decoding

In this subclause, the adaptive arithmetic coder used in octree compression is described, using the C++ style syntactic description. aa_decode() is the function which decodes a symbol using a model specified through the array cumul_freq[] and is described in the Annex G of ISO/IEC 14496-1.

5.3.1.5.1 ArithmeticDecodeSymbol

This function prepares the appropriate cumul_freq[] to be used for aa_decode function, depending on the contextID. PCT is an array of probability context tables, as described in subclause 5.3.1.6.

```
int ArithmeticDecodeSymbol (int contextID)
{
   unsigned int MAXCUM = 1<<13;
   unsigned int ImageMAXCUM = 256;
   int *p, allsym, maxcum;

   if (contextID != imageContextID)
   {
      p = PCT[contextID];
      allsym = 256;
      maxcum = MAXCUM;
   }
   else
   {
      p = TexturePCT;
      allsym = numOfImages + 1;
      maxcum = ImageMAXCUM;
   }

   int cumul_freq[allsym];
   int cum=0;
   for (int i=allsym-1; i>=0; i--)
   {
      cum += p[i];
      cumul_freq[i] = cum;
   }
   if (cum > maxcum)
   {
      cum=0;
      for (int i=allsym-1; i>=0; i--)
      {
         PCT[contextID][i] = (PCT[contextID][i]+1)/2;
         cum += PCT[contextID][i];
```

© ISO/IEC 2006 – All rights reserved

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

```
            cumul_freq[i] = cum;
        }
    }
    return aa_decode(cumul_freq);
}
```

5.3.1.6 Decoding Process

The overall structure of decoding process is described in subclause 5.3.1.4. It shows how one obtains the TBVO voxels from the stream of bits that constitute the arithmetically encoded (compressed) TBVO model.

At each step of decoding process we must update the context number (i.e. the index of probability table), and the probability table itself. The probabilistic model is the union of all probability tables (integer arrays). j-th element of i-th probability table, divided by the sum of its elements, estimate the probability of occurrence of the j-th symbol in i-th context.

The process of updating the probability table is as follows. At the start, probability tables are initialized so that all the entries are equal to 1. Before decoding a symbol, the context number (ContextID) must be chosen. ContextID is determined from previously decoded data, as indicated in 5.3.1.6.1 and 5.3.1.6.2 below. When ContextID is obtained, the symbol is decoded using binary arithmetic decoder. After that, the probability table is updated, by adding adaptive step to the decoded symbol frequency. If the total (cumulative) sum of table elements becomes greater than the cumulative threshold, then the normalization is performed (see 5.3.1.5.1).

5.3.1.6.1 Context modeling of image symbol

Image symbol is modeled with only one context (i.e. only one probability table is used). The size of this table is equal to number numOfImages plus one. At the start, this table is initialized to all ‘1’-s. The maximum allowed entry value is set to 256. The adaptive step is set to 32. This combination of parameter values allows adapting to the highly variable stream of index numbers.

5.3.1.6.2 Context modeling of voxel symbol

There are 256 different voxel symbols, each symbol representing a 2×2×2 binary voxel array. 3D orthogonal transformation may be applied to these arrays, transforming the corresponding symbols into each other.

Consider a set of 48 fixed orthogonal transforms, that is, rotations by 90*n (n=0,1,2,3) degrees about the coordinate axis, and symmetries. Their matrices are given below, in the order of their numbers:

Orthogonal Transforms [48]=

{

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix},\begin{pmatrix}0&1&0\\1&0&0\\0&0&1\end{pmatrix},\begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix},\begin{pmatrix}-1&0&0\\0&1&0\\0&0&1\end{pmatrix},\begin{pmatrix}0&1&0\\0&0&1\\1&0&0\end{pmatrix},\begin{pmatrix}0&-1&0\\1&0&0\\0&0&1\end{pmatrix},\begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix},\begin{pmatrix}-1&0&0\\0&0&1\\0&1&0\end{pmatrix},$$

$$\begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix},\begin{pmatrix}0&0&-1\\1&0&0\\0&1&0\end{pmatrix},\begin{pmatrix}0&1&0\\-1&0&0\\0&0&1\end{pmatrix},\begin{pmatrix}0&1&0\\0&0&1\\-1&0&0\end{pmatrix},\begin{pmatrix}0&-1&0\\-1&0&0\\0&0&1\end{pmatrix},\begin{pmatrix}0&0&1\\-1&0&0\\0&1&0\end{pmatrix},\begin{pmatrix}0&0&1\\0&1&0\\-1&0&0\end{pmatrix},\begin{pmatrix}0&0&-1\\-1&0&0\\0&1&0\end{pmatrix},$$

$$\begin{pmatrix}0&-1&0\\0&0&1\\1&0&0\end{pmatrix},\begin{pmatrix}0&0&-1\\0&1&0\\1&0&0\end{pmatrix},\begin{pmatrix}0&0&1\\0&-1&0\\1&0&0\end{pmatrix},\begin{pmatrix}0&0&1\\1&0&0\\0&-1&0\end{pmatrix},\begin{pmatrix}0&0&-1\\0&-1&0\\1&0&0\end{pmatrix},\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix},\begin{pmatrix}1&0&0\\0&0&1\\0&-1&0\end{pmatrix},\begin{pmatrix}-1&0&0\\0&-1&0\\0&0&1\end{pmatrix},$$

$$\begin{pmatrix}1&0&0\\0&0&-1\\0&1&0\end{pmatrix},\begin{pmatrix}1&0&0\\0&1&0\\0&0&-1\end{pmatrix},\begin{pmatrix}-1&0&0\\0&0&-1\\0&1&0\end{pmatrix},\begin{pmatrix}0&1&0\\0&0&-1\\1&0&0\end{pmatrix},\begin{pmatrix}0&1&0\\1&0&0\\0&0&-1\end{pmatrix},\begin{pmatrix}0&-1&0\\0&0&-1\\1&0&0\end{pmatrix},\begin{pmatrix}-1&0&0\\0&0&1\\0&-1&0\end{pmatrix},\begin{pmatrix}0&0&-1\\1&0&0\\0&-1&0\end{pmatrix},$$

$$\begin{pmatrix}0&0&1\\-1&0&0\\0&-1&0\end{pmatrix},\begin{pmatrix}0&0&1\\0&-1&0\\-1&0&0\end{pmatrix},\begin{pmatrix}0&0&-1\\-1&0&0\\0&-1&0\end{pmatrix},\begin{pmatrix}0&-1&0\\0&0&1\\-1&0&0\end{pmatrix},\begin{pmatrix}0&-1&0\\1&0&0\\0&0&-1\end{pmatrix},\begin{pmatrix}1&0&0\\0&-1&0\\0&0&-1\end{pmatrix},\begin{pmatrix}1&0&0\\0&-1&0\\0&0&-1\end{pmatrix},\begin{pmatrix}-1&0&0\\0&-1&0\\0&0&-1\end{pmatrix},$$

$$\begin{pmatrix}0&0&-1\\0&-1&0\\-1&0&0\end{pmatrix},\begin{pmatrix}0&0&-1\\0&1&0\\-1&0&0\end{pmatrix},\begin{pmatrix}-1&0&0\\0&-1&0\\0&0&-1\end{pmatrix},\begin{pmatrix}0&-1&0\\0&0&-1\\-1&0&0\end{pmatrix},\begin{pmatrix}0&-1&0\\-1&0&0\\0&0&-1\end{pmatrix},\begin{pmatrix}0&1&0\\0&0&-1\\-1&0&0\end{pmatrix},\begin{pmatrix}0&1&0\\-1&0&0\\0&0&-1\end{pmatrix},\begin{pmatrix}-1&0&0\\0&1&0\\0&0&-1\end{pmatrix},$$

}

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

There are 22 sets of symbols – called classes, - such that 2 symbols are connected by such a transform if and only if they belong to the same class. The coding method constructs PCT's (probability context tables) as follows: ContextID of a symbol equals either to the number of class to which its parent belongs, or to a combined number (parent class, current voxel position in the parent voxel). This allows to reduce the number of contexts greatly, and the time needed to gain meaningful statistics.

For each class, a single base symbol is determined (see Table 19), and for each symbol, the orthogonal transform that takes it into the base symbol of its class is precomputed (in actual encoding/decoding process, look-up table is used.). After the ContexID for a symbol is determined, the transform, inverse (i.e. transposed matrix) to the one taking its parent into the base element is applied.

Table 19 — An example of base symbol for each class

Class	Example of base symbol	Class order (Number of elements)
0	0	1
1	1	8
2	3	12
3	6	12
4	7	24
5	15	6
6	22	8
7	23	8
8	24	4
9	25	24
10	27	24
11	30	24
12	31	24
13	60	6
14	61	24
15	63	12
16	105	2
17	107	8
18	111	12
19	126	4
20	127	8
21	255	1

The context model depends on the number N of already decoded symbols:

For N < 512 there is only one context. The probability table is initialized to all '1'-s. The number of symbols in the probability table is 256. The adaptive step is 2. The maximum cumulative frequency is 8192.

For 512 ≤ N < 2560 (=2048+512), 1-context (in the sense that context number is single parameter, number of the class) model is used. This model uses 22 PCT's. ContextID is the number of the class to which the parent of the decoded node voxel belongs. This number can always be determined from the lookup table with base symbol and orthogonal transform, because the parent is decoded earlier than the child. Each of the 22 PCT's is initialized by the PCT from previous stage. The number of symbols in each probability table is 256. Adaptive step is 3. The maximum cumulative frequency is also 8192. After symbol is decoded, it is transformed using inverse orthogonal transform defined above.

When 2560 symbols are decoded, the decoder switches to 2-context (in the sense that the context number is now composed of the two parameters as explained below). This model uses 176 (=22*8, i.e. 22 classes by 8 positions) PCT's. ContextID here depends on the parent class and the position of the current node voxel in the parent nodevoxel. Initial probability tables for this model depend only on its context, but not position: for all 8 positions PCT is a clone of the PCT obtained for the given class at the previous stage. The number of symbols in each probability table is 256. Adaptive step is 4. The maximum cumulative frequency is also 8192.

© ISO/IEC 2006 – All rights reserved

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

After symbol is decoded it is also transformed using the inverse orthogonal transform, as is in the previous model.

In the animated case, initial context model for every frame other than the first one is inherited from the previous frame.

5.4 Compressed Bone-based animation

5.4.1 Bone-Based Animation stream encapsulation within BIFS-Anim

The Bone-based Animation (BBA) stream is connected to an **SBVCAnimation** node (see subclause 4.7.1.6) by using the BIFS-Anim mechanism (as for FBA) defined in ISO/IEC 14496-1.

The BIFS-Anim specific classes, ElemantaryMask and AnimationFrameData from subclauses 9.3.5.5 and 9.3.8.6 of ISO/IEC 14496-1, respectively, are updated as follows:

```
class ElementaryMask() {
  bit(BIFSConfiguration.nodeIDbits) nodeID;
  NodeUpdateField node = GetNodeFromID(nodeID);
  switch (node.nodeType) {
    case FaceType:
    break;
    case BodyType:
    break;
    case IndexedFaceSet2DType:
    break;
    case SBVCAnimationType:
    break;
    default:
    InitialFieldsMask initMask(node);
  }
}

class AnimationFrameData (AnimationMask mask) {
int i;
for (i=0; i<mask.numNodes; i++) {
  if (mask.isActive[i]) {
  NodeData node = mask.animNode[i]
  switch (node.nodeType) {
    case FaceType:
    FaceFrameData fdata; [ fdata ]
     break;
    case BodyType:
     BodyFrameData bdata; [bdata]
     break;
    case IndexedFaceSet2DType:
     Mesh2DframeData mdata;
    break;
    case SBVCAnimationType:
     bba_object_plane_data(); // see subclause 5.4.2.3
    break;
    default
    int j;
    for(j=0; j<node.numDYNfields; j++) {
      if (node.isAnimField[j])
AnimationField AField(node.field[node.dyn2all[j]],mask.isIntra);
     }
   }
  }
 }
}
```

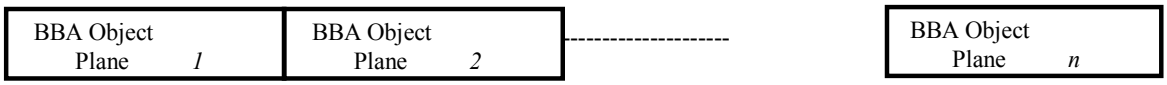

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.4.2 Bone-Based Animation stream definition

5.4.2.1 Overview

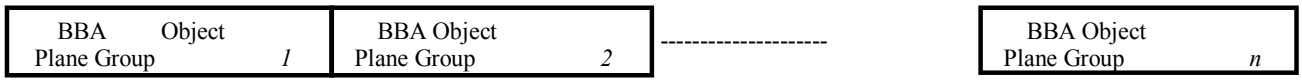
A BBA object is formed by a temporal sequence of BBA object planes as depicted below:

BBA object:

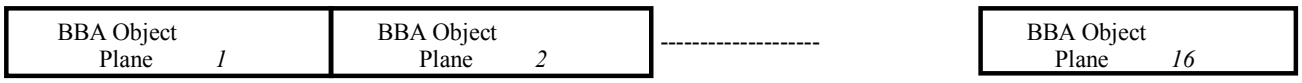


Alternatively, a BBA object can be formed by a temporal sequence of BBA object plane groups (called segments for simplicity), where each BBA object plane group itself is composed of a temporal sequence of 16 BBA object planes, as depicted in the following:

BBA object:



BBA object plane group:



When the alternative BBA object bitstream structure is employed, the bitstream is decoded by DCT-based BBA object decoding in the same manner as for the FBA stream. Otherwise, the bitstream is decoded by the frame-based BBA object decoding.

The BBA stream is an extended form of the FBA stream. See ISO/IEC 14496-2, Coding of Audio-Visual Objects:Visual for details on FBA stream. A BBA Object plane contains the update values for Skin&Bones components (SBC) which can be translations, rotation angles, scale factors in an arbitrary direction - for bones, control points translation, control points weights factors, knots - for muscles, and weights of the target meshes – for morphing.

5.4.2.2 bba_object

5.4.2.2.1 Syntax

```
class bba_object {
  bit(23)* next;
  if (next==0)
    bit(32) bba_object_start_code;
  do{
    bba_object_plane bbaObPlane();
    bit(23)* next;
  }while (!((next==0)&&(next!=bba_object_plane_start_code)));
}
```

5.4.2.2.2 Semantics

In the `bba_object`, a start code is sent to enable resynchronization. The first 23 bits are read ahead, and stored as the integer `next`.

If `next` is 0 (in other words, the first 23 bits if the `bba_object_plane` are 0), the first 32 bits of the `bba_object` shall be read and interpreted as a start code that precedes the `bba_object`.

`bba_object_start_code` = 000001EA in hexadecimal.

© ISO/IEC 2006 – All rights reserved

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

5.4.2.3 bba_object_plane

5.4.2.3.1 Syntax

```
class bba_object_plane() {
   bba_object_plane_header();
   bba_object_plane_data();
}
```

5.4.2.3.2 Semantics

The `bba_object_plane` is the access unit of the BBA stream. It contains the `bba_object_plane_header`, which specifies timing, and the `bba_object_plane_data`, which contains the data for all nodes (bones, muscles and morphs) being animated.

5.4.2.4 bba_object_plane_header

5.4.2.4.1 Syntax

```
class bba_object_plane_header() {
  bit(23)* next;
  if (next==0)
    bit(32) bba_object_plane_start_code;
  bit(1) isIntra;
  if (isIntra) {
    bit(1) isFrameRate;
    if (isFrameRate)
      FrameRate rate;
    bit(1) isTimeCode;
    if (isTimeCode)
      unsigned int(18) timeCode;
  }
  bit(1) hasSkipFrames;
  if (hasSkipFrames)
    SkipFrames skip;
}
```

5.4.2.4.2 Semantics

In the `bba_object_plane_header`, a start code may be sent at each intra or predictive frame to enable resynchronization. The first 23 bits are read ahead, and stored as the integer `next`.

If `next` is 0 (in other words, the first 23 bits if the `bba_object_plane` are 0), the first 32 bits of the `bba_object_plane` shall be read and interpreted as a start code that precedes the `bba_object_plane`.

If the boolean `isIntra` is TRUE, the current animation frame contains intra-coded values, otherwise it is a predictive frame.

In intra mode, some additional timing information is also specified. The timing information obeys the syntax of the Facial Animation specification in ISO/IEC 14496-2. Finally, it is possible to skip a number of `bba_object_plane` by using the `FrameSkip` syntax specified in ISO/IEC 14496-2.

bba_object_plane_start_code = 000001EB in hexadecimal.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.4.2.5 FrameRate

5.4.2.5.1 Syntax

```
class FrameRate {
   unsigned int(8) frameRate;
   unsigned int(4) seconds;
   bit(1) frequencyOffset;
}
```

5.4.2.5.2 Semantics

`frame_rate` is an 8-bit unsigned integer indicating the reference frame rate of the sequence.

`seconds` 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` 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).

5.4.2.6 SkipFrame

5.4.2.6.1 Syntax

```
class SkipFrame {
   int nFrame = 0;
   do {
      bit(4) number_of_frames_to_skip;
      nFrame = number_of_frames_to_skip + nFrame;
   } while (number_of_frames_to_skip == 0b1111);
}
```

5.4.2.6.2 Semantics

`number_of_frames_to_skip` 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 a skip of up to 29 frames (pattern "11111110") to be specified. If the 8-bits pattern equals "11111111", then another 4-bits word shall 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.

5.4.2.7 bba_object_plane_data

5.4.2.7.1 Syntax

```
class bba_object_plane_data() {
  bba_object_plane_mask bbaobplmsk();
  bba_object_plane_values bbaobplvls();
}
```

5.4.2.7.2 Semantics

bba_object_plane_data class contais information about the animation mask (class bba_object_plane_mask) and the animation values (class bba_object_plane_values).

© ISO/IEC 2006 – All rights reserved

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

5.4.2.8 bba_object_plane_mask

5.4.2.8.1 Syntax

```
class bba_object_plane_mask() {
  bit(5) NumberOfInterpolatedFrames; // NIF
  if (isIntra){
    bit(5) bba_quant;
    bit(3) pow2quant;
    bit(3) noOfControllerTypes;
    bit(10) NumberOfBones; //NSBB
    bit(10) NumberOfMuscles;//NSBM
    bit(10) NumberOfMorphs;//NMF
    for (bone=0;bone<NumberOfBones;bone++){
      bit(10) BoneIdentifier; //IDB
      bone_mask bnmask();
    }
    for (ms=0;ms<NumberOfMuscles;ms++){
      bit(10) MuscleIdentifier; //IDM
      bit(6) NumberControlPoints; //NCP
      bit(6) NumberKnots; //NK
      muscle_mask msmask();
    }
    for (mf=0;mf<NumberOfMorphs;mf++){
          bit(10) MorphIdentifier; //IDMF
          bit(6) NumberOfWeights; //NW
          morph_mask mfmask();
        }
    }
}
```

5.4.2.8.2 Semantics

NumberOfInterpolatedFrames **(NIF)** - 5-bit unsigned integer indicating the number of frames that have to be interpolated between the current frame and the received frame. If 0 (zero) the decoder pass the decoded frame to the animation engine.

bba_quant - a 5-bit unsigned integer used as the index to a **bba_scale** table for computing the quantisation step size of SBC values for predictive and DCT coding. If bba_object_coding_type is predictive, the value of **bba_scale** is specified in the following list:

bba_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 bba_object_coding_type is DCT this is a 5-bit unsigned integer used as the index to a **bba_scale** table for computing the quantisation step size of DCT coefficients. The value of **bba_scale** is specified in the following list:

bba_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}

pow2quant – a 3-bit unsigned integer used as an exponent for computing the quantisation step size of SBC values.

The quantisation step is obtained as q=bba_scale[bba_quant]*2^pow2quant ;

noOfControllerTypes – a 3 bits integer indicating the number of controller types in the BBA stream.

NumberOfBones (NSBB) - a 10-bit unsigned integer indicating the number of bones animated by the current frame.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NumberOfMuscles (NSBM) - a 10-bit unsigned integer indicating the number of muscles animated by the current frame.

NumberOfMorphs (NMF) – a 10 bits integer indicating the number of morph objects that are animated in the current frame.

BoneIdentifier (IDB) a 10-bit unsigned integer indicating a bone identifier; must correspond with a boneID field from a SBBone node from the scene graph.

MuscleIdentifier (IDM) a 10-bit unsigned integer indicating a muscle identifier; must correspond with a muscleID field from a SBMuscle node from the scene graph.

NumberControlPoints (NCP) A 6-bit unsigned integer indicating the number of control points of the muscle curve.

NumberKnots (NK) A 6-bit unsigned integer indicating the number of elements in the knot sequence of muscle curve.

MorphIdentifier – a 10 bits unique identifier that indicate what morph is currently animated; this value must be identical with the MorphID field from the MorphShape Node.

NumberOfWeights – a 6 bits integer indicating the number the weights of the current morph objects.

5.4.2.9 bone_mask

5.4.2.9.1 Syntax

```
class bone_mask() {
  bit(1) IsTranslation_changed;
  bit(1) marker_bit;
  if (IsTranslation_changed){
    bit(1) IsTranslationOnX_changed;
    bit(1) IsTranslationOnY_changed;
    bit(1) IsTranslationOnZ_changed;
  }
  bit(1) IsRotation_changed;
  bit(1) marker_bit;
if (IsRotation_changed){
  bit(1) isQuaternion ;
  if (!isQuaternion){
      bit(1) IsRotationOnAxis1_changed;
      bit(1) IsRotationOnAxis2_changed;
      bit(1) IsRotationOnAxis3_changed;
}else{
      bit(1) IsRotationOnQx_changed;
      bit(1) IsRotationOnQy_changed;
      bit(1) IsRotationOnQz_changed;
      bit(1) IsRotationOnQw_changed;
}
  }
  bit(1) IsScale_changed;
  bit(1) marker_bit;
  if (IsScale_changed){
    bit(1) IsScaleOnX_changed;
    bit(1) IsScaleOnY_changed;
    bit(1) IsScaleOnZ_changed;
  }
  bit(1) IsScaleOrientation_changed;
  bit(1) marker_bit;
  if (IsScaleOrientation_changed){
    bit(1) IsScaleOrientation AxisX_changed;
    bit(1) IsScaleOrientation AxisY_changed;
    bit(1) IsScaleOrientation AxisZ_changed;
```

© ISO/IEC 2006 – All rights reserved

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

```
    bit(1) IsScaleOrientation Value_changed;
  }
  bit(1) IsCenter_changed;
  bit(1) marker_bit;
  if (IsCenter_changed){
    bit(1) IsCenterOnX_changed;
    bit(1) IsCenterOnY_changed;
    bit(1) IsCenterOnZ_changed;
  }
}
```

5.4.2.9.2 Semantics

isTranslation_changed	1 bit flag indicating, for the current bone, if at least one of the translation components is affected in the current frame
isTranslationOnX_changed	1 bit flag indicating, for the current bone, if the translation on X axis is affected in the current frame
IsTranslationOnY_changed	1 bit flag indicating, for the current bone, if the translation on Y axis is affected in the current frame
isTranslationOnZ_changed	1 bit flag indicating, for the current bone, if the translation on Z axis is affected in the current frame
isRotation_changed	1 bit flag indicating, for the current bone, if at least one of the rotation components is affected in the current frame
isRotationOnAxis1_changed	1 bit flag indicating, for the current bone, if the rotation in respect with the first axis is affected in the current frame
isRotationOnAxis2_changed	1 bit flag indicating, for the current bone, if the rotation in respect with the second axis is affected in the current frame
isRotationOnAxis3_changed	1 bit flag indicating, for the current bone, if the rotation in respect with the third axis is affected in the current frame
isScale_changed	1 bit flag indicating, for the current bone, if at least one of the scale components is affected in the current frame
isScaleOnX_changed	1 bit flag indicating, for the current bone, if the scale on X axis is affected in the current frame
isScaleOnY_changed	1 bit flag indicating, for the current bone, if the scale on Y axis is affected in the current frame
isScaleOnZ_changed	1 bit flag indicating, for the current bone, if the scale on Z axis is affected in the current frame
isScaleOrientation_changed	1 bit flag indicating, for the current bone, if at least one of the scaleOrientation components is affected in the current frame
isScaleOrientationAxisX_changed	1 bit flag indicating, for the current bone, if the scaleOrientation on X axis is affected in the current frame
isScaleOrientationAxisY_changed	1 bit flag indicating, for the current bone, if the scaleOrientation on Y axis is affected in the current frame
isScaleOrientationAxisZ_changed	1 bit flag indicating, for the current bone, if the scaleOrientation on Z axis is affected in the current frame
isScaleOrientationValue_changed	1 bit flag indicating, for the current bone, if the scaleOrientation angle value is affected in the current frame
isCenter_changed	1 bit flag indicating, for the current bone, if at least one of the center components is affected in the current frame

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

isCenterOnX_changed	1 bit flag indicating, for the current bone, if the center on X axis is affected in the current frame
isCenterOnY_changed	1 bit flag indicating, for the current bone, if the center on Y axis is affected in the current frame
isCenterOnZ_changed	1 bit flag indicating, for the current bone, if the center on Z axis is affected in the current frame

According to the received mask, the decoder sets-up the so-called " bone elementary mask" which contains 16 bits corresponding to: translation in X, translation in Y, translation in Z, rotation about first axis, rotation about second axis, rotation about third axis, scale factor along X, scale factor along Y, scale factor along Z, scaleOrientation axis X, scaleOrientation axis Y, scaleOrientation axis Z, scaleOrientation angle value, center deplacement in X, center deplacement in Y, center deplacement in Z. If one of the components is updated the elementary mask for it will be 1 (one); if not is 0 (zero).

If in the **SBBone** node the bone is defined as the end-effector of a kinematics chain, the translation component from the animation stream is used as the desired location of the end-effector. The rest of the bone transformation from the animation stream, if it exists, is ignored.

5.4.2.10 Muscle_mask

5.4.2.10.1 Syntax

```
class muscle_mask() {
  bit(1) IsControlPoints_Position_changed;
  bit(1) marker_bit;
  if (IsControlPoints_Position_changed){
    for (cp=1;cp<=NCP;cp++){
      bit(1) marker_bit;
      bit(1) IsControlPoint_onX_changed[cp];
      bit(1) IsControlPoint_onY_changed[cp];
      bit(1) IsControlPoint_onZ_changed[cp];
    }
  }
  bit(1) IsControlPoints_Weight_changed;
  bit(1) marker_bit;
  if (IsControlPoints_Weight_changed){
    for (cp=1;cp<=NCP;cp++){
      bit(1) marker_bit;
      bit(1) IsControlPoint_weight[cp];
    }
  }
  bit(1) IsKnot_changed;
  bit(1) marker_bit;
  if (IsKnot_changed){
    for (k=1;k<=NK;k++){
      bit(1) marker_bit;
      bit(1) IsKnot_changed[k];
    }
  }
}
```

© ISO/IEC 2006 – All rights reserved

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

5.4.2.10.2 Semantics

isControlPoints_Position_changed	1 bit flag indicating, for the current muscle, if at least one of the curve control points position is affected in the current frame
isControlPoints_onX_changed[cp]	1 bit flag indicating, for the current muscle, if control point index cp perform a translation on X axis in the current frame
isControlPoints_onY_changed[cp]	1 bit flag indicating, for the current muscle, if control point index cp perform a translation on Y axis in the current frame
isControlPoints_onZ_changed[cp]	1 bit flag indicating, for the current muscle, if control point index cp perform a translation on Z axis in the current frame
isControlPoints_Weight_changed	1 bit flag indicating, for the current muscle, if at least one of the curve control points weigh is affected in the current frame
isControlPoints_weight_changed[cp]	1 bit flag indicating, for the current muscle, if for the control point index cp a new weigh is transmitted in the current frame
isKnot_changed	1 bit flag indicating, for the current muscle, if at least one of the elements from the knot sequence is affected in the current frame
isKnot_changed[k]	1 bit flag indicating, for the current muscle, if for the knot element index k a new value is transmitted in the current frame

According to the received mask, the decoder set up the so-called " muscle elementary mask" which contains a variable number of bits corresponding to translation on X for the first control point of the curve, translation on Y for the first control point of the curve, translation on Z for the first control point of the curve, and so on for all the points, followed by the mask for control points weigh and the mask of knots. As for the bones, if a component is updated by the current frame the "muscle elementary mask" will be 1 (one); if not is 0 (zero).

5.4.2.11 morph_mask

5.4.2.11.1 Syntax

```
class morph_mask() {
  for (w=1;w<=NW;w++){
    bit(1) marker_bit;
    bit(1) IsWeight_changed[w];
  }
}
```

5.4.2.11.2 Semantics

IsWeight_changed – a vector with the same dimention as the weights field from the MorphShape node indicating if the weight of a target shape is changed in the current animation frame.

5.4.2.12 bba_object_plane_values

5.4.2.12.1 Syntax

```
class bba_object_plane_values() {
  if (isIntra){
    bit(1) bba_object_coding_type;
    if (bba_object_coding_type==0){
      bit(1) bba_is_i_new_max;
      bit(1) bba_is_i_new_min;
```

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      bit(1) bba_is_p_new_max;
      bit(1) bba_is_p_new_min;
      bba_new_minmax bbamnmx();
      bba_i_frame bbaifr();
    }else
      bba_i_segment bbaisg();
  }else{
    if (bba_object_coding_type==0)
      bba_p_frame bbapfr();
    else
      bba_p_segment bbapsg();
  }
}
```

5.4.2.12.2 Semantics

Bba_object_coding_type	1-bit integer indicating which coding method is used. Value 0 (zero) means the coding method is "predictive coding", value 1 (one) means that the encoding method is DCT
Bba_is_i_new_max	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
Bba_is_i_new_min	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
Bba_is_p_new_max	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
Bba_is_p_new_min	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

If **Bba_is_i_new_max** is not specified the default value of 1860 is used.

If **Bba_is_i_new_min** is not specified the default value of -1860 is used.

If **Bba_is_p_new_max** is not specified the default value of 600 is used.

If **Bba_is_i_new_max** is not specified the default value of -600 is used.

5.4.2.13 bba_new_minmax

5.4.2.13.1 Syntax

```
class bba_new_minmax() {
  if (bba_is_i_new_max){
    for (sbc=1;sbc<=NUM_SBCs;sbc++){
      bit(1) marker_bit;
      if (sbc_mask[sbc])
        bit(5) bba_i_new_max[sbc];
    }
  if (bba_is_i_new_min){
    for (sbc=1;sbc<=NUM_SBCs;sbc++){
      bit(1) marker_bit;
      if (sbc_mask[sbc])
        bit(5) bba_i_new_min[sbc];
    }
  if (bba_is_p_new_max){
    for (sbc=1;sbc<=NUM_SBCs;sbc++){
      bit(1) marker_bit;
      if (sbc_mask[sbc])
        bit(5) bba_p_new_max[sbc];
    }
  if (bba_is_p_new_min){
    for (sbc=1;sbc<=NUM_SBCs;sbc++){
      bit(1) marker_bit;
```

© ISO/IEC 2006 – All rights reserved

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

```
      if (sbc_mask[sbc])
        bit(5) bba_p_new_min[sbc];
    }
}
```

5.4.2.13.2 Semantics

bba_i_new_max[one_sbc]	5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the I frame for the current Skin&Bones component (SBC)
bba_i_new_min[one_sbc]	5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the I frame for the current Skin&Bones component (SBC)
bba_p_new_max[one_sbc]	5-bit unsigned integer used to scale the maximum value of the arithmetic decoder used in the P frame for the current Skin&Bones component (SBC)
bba_p_new_max[one_sbc]	5-bit unsigned integer used to scale the minimum value of the arithmetic decoder used in the P frame for the current Skin&Bones component (SBC)

NUM_SBCs is the number of SBC in the current frame and is computed as :

$$NUM_SBCs = \sum_{bn=1}^{NSBB} c_{bn} + \sum_{ms=1}^{NSBM} (3 * NCP_{ms} + NCP_{ms} + NK_{ms}) + \sum_{mf=1}^{NMF} NW_{mf}$$

Equation 20

with $c_{bn} = \begin{cases} 16, if & isQuaternion = 0 \\ 17, if & isQuaternion = 1 \end{cases}$ is the maximum number of the components for a bone transform.

sbc_mask contains NUM_SBCs elements and is obtained by concatenation of all the “bones elementary mask”, the “muscles elementary mask” and the "morphs elementary masks", affected in the current frame.

5.4.2.14 bba_i_frame

5.4.2.14.1 Syntax

```
class bba_i_frame() {
  for (sbc=1;sbc<=NUM_SBCs;sbc++){
    if (sbc_mask[sbc])
      aa_decode aad(isbc_Q[sbc], isbc_cum_freq[sbc]);
  }
}
```

5.4.2.14.2 Semantics

The SBC are quantized and coded by a predictive coding scheme. 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 Annex G from ISO/IEC 14496-1, *Coding of Audio-Visual Objects: Systems*.

The SBCs have the same precision requirement. Two information are used to obtain the quantisation step : the bba_quant which is an index parameter ranges from 0 to 31 and is an index to a sbc_pred_scale_table :

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

int sbc_pred_scale_table [32] = { 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};

and the pow2quant which is en parameter ranges from 0 to 8.

The value of (SBC_QUANT = sbc_pred_scale_table [bba_quant]) == 0 has a special meaning, it is used to indicate lossless coding mode, so no dequantisation is applied. The quantisation stepsize is obtained as follows:

```
SBC_QUANT = sbc_pred_scale_table [bba_quant]
if (SBC_QUANT)
    qstep = 2^pow2quant * SBC_QUANT
else
    qstep = 1
```

The dequantized SBC'(t) is obtained from the decoded coefficient SBC''(t) as follows:

SBC'(t) = qstep * SBC''(t)

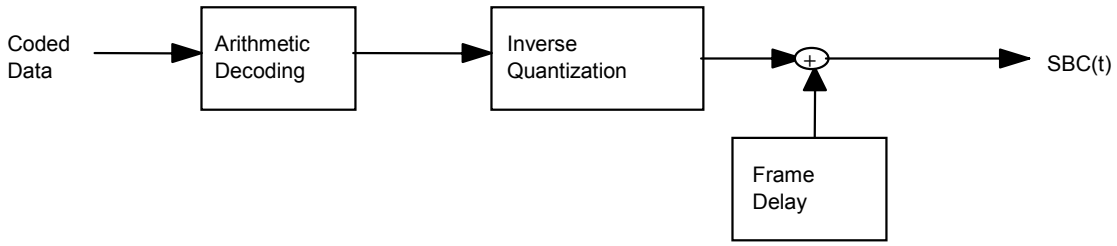


Figure 47 – SBC predictive coding.

5.4.2.15 bba_p_frame

5.4.2.15.1 Syntax

```
class bba_p_frame() {
  for (sbc=1;sbc<=NUM_SBCs;sbc++){
    if (sbc_mask[sbc])
      aa_decode aad(psbc_diff[sbc], psbc_cum_freq[sbc]);
  }
}
```

5.4.2.15.2 Semantics

See subclause 5.4.2.14.2.

5.4.2.16 bba_i_segment

5.4.2.16.1 Syntax

```
class bba_i_segment() {
  for (sbc=1;sbc<=NUM_SBCs;sbc++){
    if (sbc_mask[sbc]){
      decode_i_dc didc(dc_Q[sbc]);
      decode_ac   dac(ac_Q[sbc]);
    }
  }
}
```

© ISO/IEC 2006 – All rights reserved

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

5.4.2.16.2 Semantics

The bitstream is decoded into segments of SBCs, where each segment is composed of a temporal sequence of 16 SBCs object planes. The block diagram of the decoder is the following:

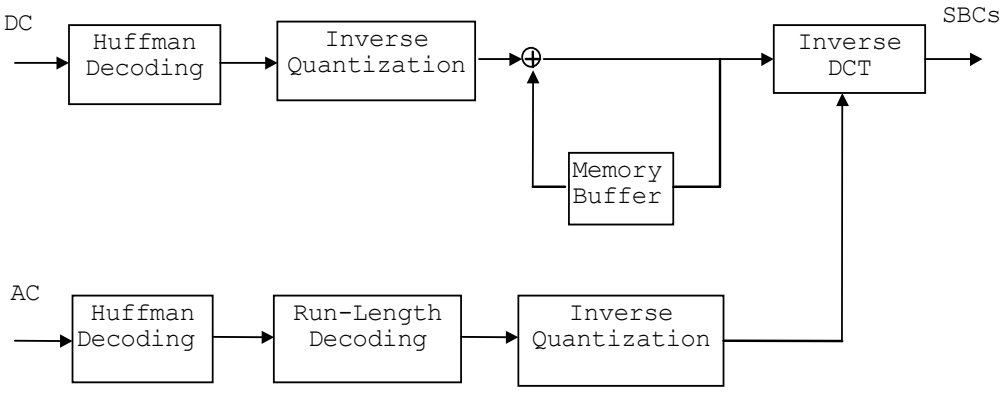


Figure 48 – Block diagram of the DCT-based SBCs decoding process.

The DCT-based decoding process consists of the following three basic steps:

i) Differential decoding the DC coefficient of a segment.

j) Decoding the AC coefficients of the segment

k) Determining the 16 SBCs 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]  = sbc_scale[bba_quant] * 2^pow2quant ;
```

with

```
static int sbc_scale [32] = { 0, 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, 1023 };
```

The quantisation step size of the DC coefficient is one-third of the AC coefficients.

The DCT-based decoding process is applied to all SBCs. 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.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.4.2.17 bba_p_segment

5.4.2.17.1 Syntax

```
class bba_p_segment() {
  for (sbc=1;sbc<=NUM_SBCs;sbc++){
    if (sbc_mask[sbc]){
      decode_p_dc dpdc(dc_Q[sbc]);
      decode_ac   dac(ac_Q[sbc]);
    }
  }
}
```

5.4.2.17.2 Semantics

See subclause 5.4.2.16.

5.4.2.18 decode_i_dc

5.4.2.18.1 Syntax

```
class decode_i_dc(dc_q) {
  bit(16) dc_q;
  if (dc_q ==-256*128)
    bit(32) dc_q;
}
```

5.4.2.18.2 Semantics

See subclause 5.4.2.16.

5.4.2.19 decode_p_dc

5.4.2.19.1 Syntax

```
class decode_p_dc(dc_q_diff) {
  dc_q_diff; // decode Huffman
  dc_q_diff=dc_q_diff-256;
  if (dc_q_diff==-256)
    bit(16) dc_q_diff;
  if (dc_Q == 0-256*128)
    bit(32) dc_q_diff;
}
```

5.4.2.19.2 Semantics

See subclause 5.4.2.16.

5.4.2.20 decode_ac

5.4.2.20.1 Syntax

```
class decode_ac(ac_q[]) {
  this=0;
  next=0;
  while (next<15){
    bit(4) count_of_runs;
    if (count_of_runs == 15)
      next = 16;
    else {
```

© ISO/IEC 2006 – All rights reserved

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

```
      next=this+1+count_of_runs;
      for(n=this+1;n<next;n++)
        ac_q[i][n]=0;
      bit(8) ac_q[i][next];
      if (ac_q[i][next]==256)
        decode_i_dc didc(ac_q[i][next]);
      else
        ac_q[i][next]-=256;
      this=next;
    }
  }
}
```

5.4.2.20.2 Semantics

See subclause 5.4.2.16.

5.5 AFX Generic Backchannel

5.5.1 Usage of the backchannel

The backchannel can be used by the following tools : VTC textures, WaveletSubdivisionSurfaces, MeshGrid.

In the first case, i.e. VTC textures, the backchannel will be an elementary stream of stream-type ‘visual’ (0x04) and with an ObjectTypeIndication 0x20.

In the case of WaveletSubdivisionSurfaces and MeshGrid, the backchannel is an elementary stream of stream type ‘scene description’ (0x03) with an OTI 0x05. The AFXConfig contains the appropriate code to make the distinction between AFX tools (see ISO/IEC 14496-1).

Each backchannel is related to a downstream elementary stream carrying the data associated with each tool.

This means that AFX tools share the same bitstream syntax but each instance has a separate elementary stream.

The messages carried in an AccessUnit of a backchannel consist of a list of BackChannelCommands as specified below. The syntax of an access unit is defined as follows.

5.5.2 BackChannel Access Unit

5.5.2.1 Syntax

```
class BackChannelAccessUnit {
   BackChannelCommand[0..255] backchannelCommands;
}
```

5.5.2.2 Semantics

An access unit in the back channel is an array of BackChannelCommands.

5.5.3 BackChannelCommand

5.5.3.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class BackChannelCommand : bit(8) tag=0 {
   // empty. To be filled by classes extending this class.
}
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.5.3.2 Semantics

The possible values of BackChannelCommand tag and their interpretation is given in Table 20.

Table 20 — List of Class Tags for Backchannel Commands

Tag value	Tag name
0x00	Forbidden
0x01	GlobalParametersTag
0x02	NavigationParametersTag
0x03	ToolGenericBackChannelTag
0x04-0xBF	Reserved for ISO (backchannel command tags)
0xC0-0xFE	User private
0xFF	Forbidden

5.5.3.3 General bitstream syntax.

The bitstream is an array of BackChannelCommands.

The server decodes all backchannelcommands of the access unit which it can parse, otherwise it skips them.

The configuration of the stream describes the list of tags are mandatory in the bitstream as well as those that are optional (i.e. that a server may parse). Other backchannel commands will be skipped by a compliant server.

5.5.3.4 Definitions

5.5.3.4.1 Client/Server-driven scenario's.

When considering view-dependent 3D transmission, two possible scenarios occur. Either the client tracks the cache status and asks the server for the not-yet-transmitted information, either it is the server that gets the client's cache status back and calculates out of this which additional information to transmit for the current viewpoint. Said in other words: the control about what to transmit and what is already available is either done by the client (first scenario), either by the server (second scenario). The scenarios are therefore called respectively *client-driven* and *server-driven*.

In a *client-driven* scenario, neither the viewpoint, nor the cache status have to be transmitted, since this information is used by the client only.

In the *server-driven* scenario, both the viewpoint and cache status have to be transmitted to the server.

5.5.3.4.2 Unit.

A *unit* is a generic term that for each tool may have one or several meanings depending on the context as follows:

1. For wavelet subdivision surface: units represent hierarchical trees corresponding to small regions of the base mesh.

2. For VTC two contexts exist:

    a. A unit represents a Texture Unit that can be extracted from a packet in VTC's error resilience mode.

    b. A unit represents an ROI corresponding to a block in the spatial domain of size $2^{levels}x2^{levels}$ pixels. In the wavelet domain this corresponds to 9 trees (for each Y, U and V color

© ISO/IEC 2006 – All rights reserved

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

component: the low-high, high-low and high-high subimage tree) or 3 trees (when only the Y component applies). The numbering of these ROIs is row-wise, starting with number 1 as shown in the following picture:

ROI nr 1	ROI nr 2	ROI nr 3	ROI nr 4
ROI nr 5	ROI nr 6	ROI nr 7	ROI nr 8
ROI nr 9	ROI nr 10	ROI nr 11	ROI nr 12
ROI nr 13	ROI nr 14	ROI nr 15	ROI nr 16

3. For MeshGrid two contexts are available:

   a. In mesh related commands (MESH_HEADER, MESH_CONNECTIVITY, VERTICES_REPOSITION and VERTICES_REFINEMENT) a unit represents a region of interest (ROI).

   b. In grid related commands (GRID_HEADER and GRID_COEFFICIENTS) a unit represents a tile.

5.5.3.5 GlobalParameters

5.5.3.5.1 Syntax.

```
class GlobalParameters extends BackChannelCommand:
bit(8) tag = GlobalParametersTag {
   SFVec2f FieldOfView;
   SFFloat NearPlane;
   SFFloat FarPlane;
   SFFloat VisibilityLimit;
}
```

5.5.3.5.2 Semantics.

Field of view: these are the horizontal and vertical angles of the view pyramid.

NearPlane: this is the distance to the *near plane* of the viewing pyramid, i.e. a plane parallel to the projection plane and at distance **NearPlane** to the observer. Every point on the same side as the observer is supposed to be non visible.

FarPlane: this is the distance to the *far plane* of the viewing pyramid, i.e. a plane parallel to the projection plane and at distance **FarPlane** to the observer. Every point beyond is non visible.

VisibilityLimit: this is a float indicating a distance to the observer beyond which the scene does not have to be displayed. This might be useful, e.g., for simulating fog.

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.5.3.6 NavigationParameters

5.5.3.6.1 Syntax.

```
class NavigationParameters extends BackChannelCommand:
bit(8) tag = NavigationParametersTag {
   SFVec3f Position;
   SFRotation Direction;
}
```

5.5.3.6.2 Semantics.

Position: this is the observer position in the coordinate system of the node corresponding to the transmitted object.

Direction: this is the observer orientation in the coordinate system of the node corresponding to the transmitted object.

5.5.3.7 ToolGenericBackChannel

5.5.3.7.1 Syntax

```
class ToolGenericBackChannel extends BackChannelCommand:
bit(8) tag = ToolGenericBackChannelTag {
   bit (1) isClientDriven;
   if (!isClientDriven) {
      bit (2) cacheMode;
   }
   ListOfUnits listOfUnits[];
}
```

5.5.3.7.2 Semantics

isClientDriven: 1-bit flag specifying if equal to '1' that the backchannel is in *client-driven* scenario and conveys client requests. If the value is equal to '0' the backchannel is in *server-driven* scenario and the client sends status information to the server. The requests and status information are specific for each tool extending the generic ToolGenericBackChannel class.

cacheMode: 2-bit value specifying the type and format of the status information send in the *server-driven* scenario. The following table shows the possible values and meanings of cacheMode. In **DELTA_CACHE** mode the status information contains details about received units that have been totally or partially discarded. Both the **RECEIVED_CACHE** and **FULL_CACHE** modes reflect the same type of cache information, however using a different format. In **RECEIVED_CACHE** mode the units are explicitly identified by an index. The status information expressed using this format is compact if consecutive units in the clients cache have the same "quality" parameters and thus can be grouped together by means of **ParseIndices** class. In the opposite case when consecutive units in the client's cache have different "quality" parameters, it is more efficient to send the status information in **FULL_CACHE** mode by iterating and querying the entire range of units.

cacheMode	Meaning
0x0	**DELTA_CACHE:** Delta cache status, i.e. received units that have been partially or totally discarded.
0x1	**RECEIVED_CACHE:** Cache status, i.e. information only about the received units.
0x2	**FULL_CACHE:** Full cache status, i.e. information about the entire range of units.
0x3	Reserved

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

© ISO/IEC 2006 – All rights reserved

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

listOfUnits: Specifies an array of commands that have to be processed simultaneously, in order to have a consistent decoding process at the client. In particular, if a Region of Interest (ROI) has to be decoded at a given quality, all data that contributes to this ROI (through data dependencies) should be decoded properly. This may require several, successive commands, each pertaining to specific information to be decoded at a specific quality.

5.5.3.8 ListOfUnits

5.5.3.8.1 Syntax

```
abstract aligned(8) expandable(2^28-1) class ListOfUnits (bool isClientDriven, unsigned int
cacheMode) : bit(8) tag=0 {
   // empty. To be filled by classes extending this class.
}
```

5.5.3.8.2 Semantics

This is an abstract base class for the different types of command units in the backchannel stream. This class is extended by the classes identified by the class tags defined in the following table.

Table 21 – List of Class Tags for Descriptors derived from ListOfUnits.

Tag value	Tag name	Description
0x00	Forbidden	
0x01	ListOfVTCUnitsTag	Tag for VTC units
0x02	ListOfWaveletTreesTag	Tag for WaveletSubdivisionSurface units
0x03	MGFullStreamTag	Tag for the entire MeshGrid stream.
0x04	MGMeshDescrTag	MeshGrid stream tag for the mesh coding information at specified mesh resolution level.
0x05	MGGridDescrTag	MeshGrid stream tag for grid coding information at specified grid resolution level.
0x06	MGMeshConnectivityDescrTag	MeshGrid stream tag for mesh connectivity information at specified mesh resolution level.
0x07	MGVerticesRepositionDescrTag	MeshGrid stream tag for vertices' reposition bits (single bit-plane) at specified mesh resolution level.
0x08	MGVerticesRefinementDescrTag	MeshGrid stream tag for refinement bit-planes (the offset) at specified mesh resolution level.
0x09	MGGridCoefficientsDescrTag	MeshGrid stream tag for wavelet coefficients at specified grid resolution level.
0x10	MGGridCornersDescrTag	MeshGrid stream tag for the grid corners.
0x11-0xFE	Reserved for ISO use	
0xFF	Forbidden	

5.5.3.9 GetValue

5.5.3.9.1 Syntax

```
class GetValue(BYTE nBits)
{
   int (nBits) noBits;
   int (noBits) value;
}
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.5.3.9.2 Semantics

The **GetValue** class parses a variable length integer *value*. The result is returned in *value*.

5.5.3.10 ParseIndices

5.5.3.10.1 Syntax

```
class ParseIndices(BYTE nBits) {
   unsigned int i = 0, j = 0, k = 0;

   // the number of elements in the indices list.
   GetValue number(nBits);

   for (i = 0; i < number.value; i++) {
      bit (1) isIndexed;
      if (isIndexed) {
         GetValue unit[[k++]](nBits);    // index of the unit
      } else {
         GetValue start(nBits);          // start index of the unit
         GetValue count(nBits);          // number of indices in the range
         for (j = 0; j < count.value; j++) {
            unit[[k++]] = start.value + j;
         }
      }
   }

   // assign the total number of units.
   number.value = k;
}
```

5.5.3.10.2 Semantics

Parse a list of units that can be read one by one with an index and/or units that can be read as a successive range.

number: The initial parsed value represents the number of elements in the indices list. A list element can be the index of a unit (**unit**), or the start index (**start**) of a range of indices followed by a number (**count**) specifying the number of indices in the range. The final value of number is to the total number of units from the **unit** array.

isIndexed: Flag, for which if equal to '1' specifies that the following list element is an unit index; otherwise it specifies that the following two values represent the start index of a unit (**start**) and the number of indices in the range (**count**).

unit: Array of size **number** consisting of indices of units.

5.5.3.11 ListOfVTCUnit*s*

5.5.3.11.1 Syntax

```
class ListOfVTCUnits extends ListOfUnits(bool isClientDriven, unsigned int cacheMode):
bit(8) tag = ListOfVTCUnitsTag {
   bit (1) isROI;
   int i,j = 0;
   if (isROI)
   {
      if (isClientDriven) {
         int (5) firstWantedLevel;
         int (5) lastWantedLevel;
         int (5) firstWantedBitplane;
         int (5) lastWantedBitplane;
```

© ISO/IEC 2006 – All rights reserved

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

```
        ParseIndices ROI(5);
      } else if (cacheMode == DELTA_CACHE || cacheMode == RECEIVED_CACHE) {
        int (5) lastCachedLevel;
        int (5) lastCachedBitplane;
        ParseIndices ROI(5);
      } else {                  // server-driven & cacheMode == FULL_CACHE
        GetValue numberOfROIs(5);
        for (j=0; j<numberOfROIs.value; j++) {
          bit(1) isInCache;
          if (isInCache) {
            int (5) lastCachedLevel;
            int (5) lastCachedBitplane;
          }
        }
      }
    }
    else
    {
      if (isClientDriven || cacheMode == DELTA_CACHE || cacheMode == RECEIVED_CACHE) {
        ParseIndices TU(5);
      } else {                // server-driven & cacheMode == FULL_CACHE
        GetValue numberOfTUs(5);
        for (j=0; j<numberOfTUs.value; j++) {
          bit(1) isInCache;
        }
      }
    }
}
```

5.5.3.11.2 Semantics

In VTC, the quality of decoded texture regions can be set by selecting the number of decoded wavelet levels and bitplanes.

isROI: this bit indicates the type of units used by the command. If true, ROI numbers are used with specified wavelet levels and bitplanes. If false, texture units (TU) are used. The numbering of the TUs implicitly and uniquely indicates the corresponding texture region location in spatial domain at a certain spatial/resolution layer (i.e. number of wavelet levels) and SNR layer (i.e. number of bitplanes), therefore wavelet levels and bitplanes do not have to be specified.

In the *client-driven* scenario and when isROI is true, several ROIs with the same "quality" requests (**firstWantedLevel**, **lastWantedLevel, firstWantedBitplane, lastWantedBitplane**) are grouped together in one command. When isROI is false, TUs are grouped together in one command.

firstWantedLevel, lastWantedLevel: these are the wavelet decomposition levels (ranging from first to last) that the client asks for, in order to increase the resolution-driven quality from previously cached information (which is equivalent to lastCachedLevel).

firstWantedBitplane, lastWantedBitplane: these are the bitplanes (ranging from first to last) that the client asks for, in order to increase the SNR-driven quality from previously cached information (which is equivalent to lastCachedBitplane).

In the *server-driven* scenario when the cache mode is DELTA_CACHE or RECEIVED_CACHE and when isROI is true, several ROIs with the same "quality" status (**lastCachedLevel, lastCachedBitplane**) are grouped together in a command. When isROI is false, TUs are grouped together in one command.

lastCachedLevel: this is the last wavelet decomposition level that is cached at the client side.

lastCachedBitplane: this is the last bitplane that is cached at the client side.

In the *server-driven* scenario when the cache mode is FULL_CACHE and isROI is true, iterate for all ROIs (**numberOfROIs**). For each ROI, the **isInCache** flag specifies with '1' that cache status information (**lastCachedLevel, lastCachedBitplane**) is available; conversely if **isInCache** flag is '0' then the

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

corresponding ROI is not present in the cache. When isROI is false iterate for all TUs. For each TU, the **isInCache** flag specifies with '1' that the TU is available; conversely if **isInCache** flag is '0' then the corresponding TU is not present in the cache.

5.5.3.12 ListOfWaveletTrees

5.5.3.12.1 Syntax

```
class ListOfWaveletTrees extends ListOfUnits(bool isClientDriven, unsigned int cacheMode):
bit(8) tag = ListOfWaveletTreesTag {
   int i = 0;
   if (isClientDriven) {
      GetValue numWaveletCoefficients(5);
      int(5) firstWantedBitplane;
      int(5) lastWantedBitplane;
      ParseIndices tree(5);
   } else if (cacheMode == DELTA_CACHE || cacheMode == RECEIVED_CACHE) {
      GetValue numWaveletCoefficients(5);
      int(5) lastCachedBitPlane;
      ParseIndices tree(5);
   } else {                  // server-driven & cacheMode == FULL_CACHE
      GetValue numberOfTrees(5);
      for (i=0; i<numberOfTrees.value; i++) {
         bit(1) isInCache;
         if (isInCache) {
            GetValue numWaveletCoefficients(5);
            int(5) lastCachedBitPlane;
         }
      }
   }
}
```

5.5.3.12.2 Semantics

The units for WaveletSubdivisionSurfaces are trees used in the SPIHT representation of the wavelet coefficients.

NumWaveletCoefficients: this is the number of wavelet coefficients. If **isClientDriven** value is 1, **numWaveletCoefficients** is understood as the number of coefficients the client needs for tree number i. If **isClientDriven** value is 0, **numWaveletCoefficients** is understood as the number of coefficients that the client cache contains for ROI number i (if **isClientDriven** value is 1) or number **TreeNb** (if **isClientDriven** value is 0 and **isCacheDelta** value is 1).

firstWantedBitplane: this is the number of the first requested bitplane.

lastWantedBitplane: this is the number of the last requested bitplane.

lastCachedBitPlane: if **cacheMode** value is DELTA_CACHE or RECEIVED_CACHE, this is the number of bitplanes present in the cache for each tree contained in tree; if **cacheMode** value is FULL_CACHE, this is the number of bitplanes present in the cache for tree number i.

tree: if **isClientDriven** has value 1, this is the sequence of trees to which **NumWaveletCoefficients**, **firstWantedBitplane** and **lastWantedBitplane** refer. Otherwise this is the sequence of trees to which **NumWaveletCoefficients** and **lastCachedBitplane** refer.

NumberOfTrees: this is the total number of trees for the object.

IsInCache: this is a flag with value 1 if and only if cache information for tree number i is following.

© ISO/IEC 2006 – All rights reserved

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

5.5.3.13 MGFullStreamCommand

5.5.3.13.1 Syntax

```
class MGFullStreamCommand extends ListOfUnits(bool isClientDriven, unsigned int
cacheMode): bit(8) tag= MGFullStreamTag {
   // empty
}
```

5.5.3.13.2 Semantics

For both *client-driven* and *server-driven* scenarios request the entire stream in one step. This command is equivalent with grouping in one request several commands issued for each resolution level of the mesh, respectively grid, but it is far more compact.

5.5.3.14 MGLevelCommand

5.5.3.14.1 Syntax

```
abstract class MGLevelCommand extends ListOfUnits(bool isClientDriven, unsigned int
cacheMode): bit(8) tag=0
{
   // read the variable length counter sizeOfInstance
   unsigned int(LEVEL_BITS) resolutionLevel;
}
```

5.5.3.14.2 Semantics

This is an abstract class that serves as a base class for MeshGrid related command units. MGLevelCommand parses the resolution level (*resolutionLevel*) of the unit.

5.5.3.15 MGLayerCommand

5.5.3.15.1 Syntax

```
abstract class MGLayerCommand extends MGLevelCommand(bool isClientDriven, unsigned int
cacheMode): bit(8) tag=0
{
   // read the variable length counter sizeOfInstance
   ParseIndices identifier(nBitsLayer);
}
```

5.5.3.15.2 Semantics

This is an abstract class that serves as a base class for MeshGrid related command units. MGLayerCommand parses an *identifier* of the surface layers the MeshGrid stream unit refers to. When the value of *identifier.number* is ‘0’ then the request is is generic for all surface layers.

5.5.3.16 MGMeshDescriptorCommand

5.5.3.16.1 Syntax

```
class MGMeshDescriptorCommand extends MGLayerCommand(bool isClientDriven, unsigned int
cacheMode): bit(8) tag= MGMeshDescrTag {
   // empty
}
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.5.3.16.2 Semantics

For both *client-driven* and *server-driven* scenarios request mesh coding information for a specified mesh resolution level (**resolutionLevel**).

5.5.3.17 MGGridDescriptorCommand

5.5.3.17.1 Syntax

```
class MGGridDescriptorCommand extends MGLevelCommand(bool isClientDriven, unsigned int
cacheMode): bit(8) tag= MGGridDescrTag {
   // empty
}
```

5.5.3.17.2 Semantics

For both *client-driven* and *server-driven* scenarios request grid coding information at a specified grid resolution level (**resolutionLevel**).

5.5.3.18 MGMeshConnectivityCommand

5.5.3.18.1 Syntax

```
class MGMeshConnectivityCommand extends MGLayerCommand(bool isClientDriven, unsigned int
cacheMode): bit(8) tag= MGMeshConnectivityDescrTag {
   if (isClientDriven ||
       cacheMode == DELTA_CACHE || cacheMode == RECEIVED_CACHE) {
      ParseIndices roi(ROI_BITS);
   } else {                // server-driven & cacheMode == FULL_CACHE
      GetValue numberOfROIs(ROI_BITS);
      for (j = 0; j < numberOfROIs.value; j++) {
         bit(1) isInCache;
      }
   }
}
```

5.5.3.18.2 Semantics

The mesh connectivity information for specified regions of interest (ROIs) and mesh resolution level (**resolutionLevel**) is requested in *client-driven* mode, respectively the corresponding cache status is sent in *server-driven* mode. The ROIs are given explicitly in client-driven scenario, or in server-driven scenarios with DELTA_CACHE and RECEIVED_CACHE operation modes. In server-driven scenario FULL_CACHE operation mode all possible ROIs (**numberOfROIs**) are iterated, and for each ROI the **isInCache** flag specifies if '1' that the connectivity description is available at the client site, and if '0' it indicates the opposite.

5.5.3.19 MGVerticesRepositionCommand

5.5.3.19.1 Syntax

```
class MGVerticesRepositionCommand extends MGLayerCommand(bool isClientDriven, unsigned int
cacheMode) : bit(8) tag= MGVerticesRepositionDescrTag {
   if (isClientDriven ||
       cacheMode == DELTA_CACHE || cacheMode == RECEIVED_CACHE) {
      ParseIndices roi(ROI_BITS);
   } else {                // server-driven & cacheMode == FULL_CACHE
      GetValue numberOfROIs(ROI_BITS);
      for (j=0; j<numberOfROIs.value; j++) {
         bit(1) isInCache;
      }
   }
}
```

© ISO/IEC 2006 – All rights reserved

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

5.5.3.19.2 Semantics

The vertices' reposition bit-plane for specified regions of interest (ROIs) and mesh resolution level (**resolutionLevel**) is requested in *client-driven* mode, respectively the corresponding cache status is sent in *server-driven* mode. The ROIs are given explicitly in client-driven scenario, or in server-driven scenarios with DELTA_CACHE and RECEIVED_CACHE operation modes. In server-driven scenario FULL_CACHE operation mode all possible ROIs (**numberOfROIs**) are iterated, and for each ROI the **isInCache** flag specifies if '1' that the vertices' reposition bit-plane is available at the client site, and if '0' it indicates the opposite.

5.5.3.20 MGVerticesRefinementCommand

5.5.3.20.1 Syntax

```
class MGVerticesRefinementCommand extends MGLayerCommand(bool isClientDriven, unsigned int
cacheMode): bit(8) tag= MGVerticesRefinementDescrTag {
   if (isClientDriven) {
      bit(REFINE_BITS) firstWantedBitPlane;
      bit(REFINE_BITS) lastWantedBitPlane;
      ParseIndices roi(ROI_BITS);
   } else if (cacheMode == DELTA_CACHE || cacheMode == RECEIVED_CACHE) {
      bit(REFINE_BITS) lastCachedBitPlane;
      ParseIndices roi(ROI_BITS);
   } else {                // server-driven & cacheMode == FULL_CACHE
      GetValue numberOfROIs(ROI_BITS);
      for (j=0; j<numberOfROIs.value; j++) {
         bit(1) isInCache;
         if (isInCache) {
            bit(REFINE_BITS) lastCachedBitPlane;
         }
      }
   }
}
```

5.5.3.20.2 Semantics

For specified regions of interest (ROIs) and mesh resolution level (**resolutionLevel**), the vertices' refinement bit-planes, starting with **firstWantedBitPlane** and ending with **lastWantedBitPlane,** are requested in *client-driven* scenario, respectively the corresponding last bit-plane available in the cache (**lastCachedBitPlane**) is sent in the *server-driven* scenario. The ROIs are given explicitly in client-driven scenario, or in server-driven scenarios with DELTA_CACHE and RECEIVED_CACHE operation modes. In server-driven scenario FULL_CACHE operation mode all possible ROIs (**numberOfROIs**) are iterated, and for each ROI the **isInCache** flag specifies if '1' that the connectivity description is available at the client site, and if '0' it indicates the opposite.

5.5.3.21 MGGridCoefficientsCommand

5.5.3.21.1 Syntax

```
class MGGridCoefficientsCommand extends MGLevelCommand(bool isClientDriven, unsigned int
cacheMode): bit(8) tag= MGGridCoefficientsDescrTag {
   if (isClientDriven) {
      bit(FIELD_BITS) firstWantedBitPlane;
      bit(FIELD_BITS) lastWantedBitPlane;
      ParseIndices tile(ROI_BITS);
   } else if (cacheMode == DELTA_CACHE || cacheMode == RECEIVED_CACHE) {
      bit(FIELD_BITS) lastCachedBitPlane;
      ParseIndices tile(ROI_BITS);
   } else {                // server-driven & cacheMode == FULL_CACHE
      GetValue numberOfTiles(ROI_BITS);
      for (j = 0; j < numberOfTiles.value; j++) {
         bit(1) isInCache;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        if (isInCache) {
          bit(FIELD_BITS) lastCachedBitPlane;
        }
      }
    }
}
```

5.5.3.21.2 Semantics

At any grid resolution level (**resolutionLevel**) for any encoded tile (**tile**, **numberOfTiles**) there are three binary streams, called *component streams*, each of them corresponding to one of the {*x*,*y*,*z*} wavelet encoded coordinates of the grid points. In *client-driven* scenario, the grid coding bitplanes, starting with **firstWantedBitPlane** and ending with **lastWantedBitPlane,** are request to the server. In *server-driven* scenario, the client sends the last bit-plane available in the cache (**lastCachedBitPlane**). In *client-driven* scenario, and in *server-driven* scenarios with DELTA_CACHE and RECEIVED_CACHE operating modes, the tiles (**tile**) are explicitly specified using ParseIndices. In *server-driven* scenario FULL_CACHE operation mode, all the tiles (**numberOfTiles**) are iterated, and for each tile the **isInCache** flag specifies if '1' that cache information is available at the client site, and if '0' it indicates the opposite.

5.5.3.22 MGGridCornersCommand

5.5.3.22.1 Syntax

```
class MGGridCornersCommand extends ListOfUnits(bool isClientDriven, unsigned int
cacheMode): bit(8) tag=MGGridCornersDescrTag
{
   // empty
}
```

5.5.3.22.2 Semantics

In *client-driven* scenario request, respectively in *server-driven* scenario send, the coordinates of the reference-grid corners.

5.6 PointTexture stream

5.6.1 Overview

The PointTexture compression is a tool to compress the PointTexture node efficiently. The decoder structure of the PointTexture compression is shown in Figure 49. The PointTexture decoder consists of header decoder and node decoder. The header information is decoded in the header decoder and is used in node decoder. The PointTexture decoder receives the arithmetic coded bitstream and restores the PointTexture node. The decoded PointTexture node specified in subclause 4.5.2.3 has the depth information and the color information.

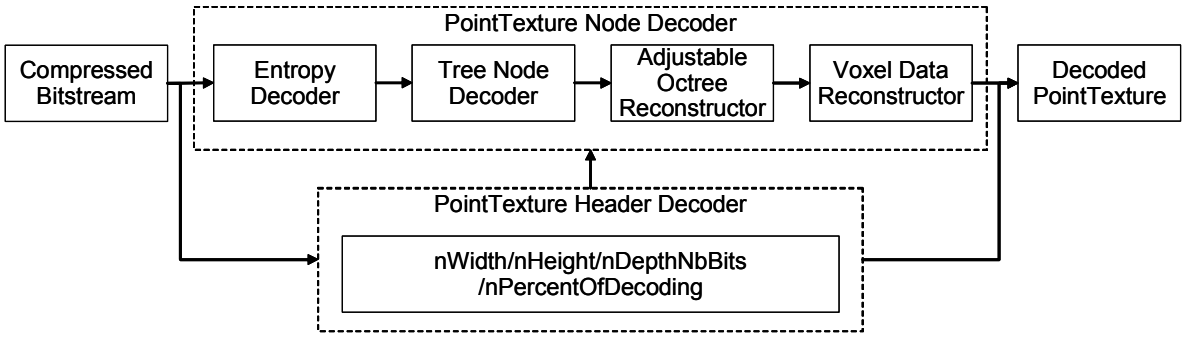


Figure 49 — Block diagram of decoder for PointTexture compression.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.6.2 PointTexture class

5.6.2.1 Syntax

```
class PointTexture ()
{
   PointTextureHeader ();
   PointTextureTreeNodes ();
}
```

5.6.2.2 Semantics

This is a top class for reading the compressed bitstream of PointTexture. PointTextureHeader is the class for reading header information from the bitstream. PointTextureTreeNodes is the class for reading tree node information progressively from low to high resolution.

5.6.3 PointTextureHeader class

5.6.3.1 Syntax

```
class PointTextureHeader ()
{
   unsigned int(5) nBitSizeOfWidth;
   unsigned int(nBitSizeOfWidth) nWidth;
   unsigned int(5) nBitSizeOfHeight;
   unsigned int(nBitSizeOfHeight) nHeight;
   unsigned int(5) nDepthNbBits;
   unsigned int(7) nPercentOfDecoding;
}
```

5.6.3.2 Semantics

nBitSizeOfWidth: This value indicates the bit size of nWidth.

nWidth: This value indicates the width of the PointTexture.

nBitSizeOfHeight: This value indicates the bit size of nHeight.

nHeight: This value indicates the height of the PointTexture.

nDepthNbBits: This value indicates the number of bits used for representing the original depth data. The value of nDepthNbBits ranges from 0 to 31, and the number of bits used in the original data is nDepthNbBits + 1.

nPercentOfDecoding: This value indicates the percent of the tree nodes to be decoded. If the value is the maximum (100), the lossless decoding is performed. Otherwise, the lossy decoding is performed.

5.6.4 PointTextureTreeNodes class

5.6.4.1 Syntax

```
class PointTextureTreeNodes ()
{
   nNumberOfTreeNodes = initializeOctree(nWidth, nHeight, nDepthNbBits);
   nNumberLimit = nNumberOfTreeNodes * nPercentOfDecoding / 100;
   pushQ(0);  // 0: root
   nCount = 0;

   while(nCount < nNumberLimit)
   {
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      if(isQueueEmpty() == true)   // break if queue is empty
         break;

      nIndex = popQ();
      nCount++;

      nSOP = decodeAAC(contextSOP);
      if(nSOP == 0)   // Split node decoding
      {
         nRegionRed = decodeAAC(contextRedOfRegion);
         nRegionGreen = decodeAAC(contextGreenOfRegion);
         nRegionBlue = decodeAAC(contextBlueOfRegion);
         for(nChild = 1; nChild <= 8; nChild++)   // 8 children nodes
         {
            nBOW = decodeAAC(contextBOW);   // black or white
            if(nBOW == 0)  // 0: white node
               nCount += getCountOfTreeSize(nIndex*8+nChild);
            else   // 1: black node
               pushQ(nIndex*8+nChild);
         }
      }
      else   // PPM node decoding
      {
         getRegion(nIndex, nStartX, nStartY, nStartZ, nEndX, nEndY, nEndZ);
         for(k = nStartZ; k < nEndZ; k++)
         {
            for(j = nStartY; j < nEndY; j++)
            {
               for(i = nStartX; i < nEndX; i++)
               {
                  nIndexOfContext = getIndexOfContext(i, j, k);
                  nVoxel = decodeAAC(contextTreeNodes[nIndexOfContext]);
                  if(nVoxel == 1)  // 1: black node
                  {
                     nDeltaRed = decodeAAC(contextColorDifference);
                        nDeltaGreen = decodeAAC(contextColorDifference);
                        nDeltaBlue = decodeAAC(contextColorDifference);
                  }
               }
            }
         }
         nCount += getCountOfTreeSize(nIndex) - 1;
      }
   }
}
```

5.6.4.2 Semantics

nNumberOfTreeNodes: This value indicates the number of the tree nodes in an octree.

initializeOctree: This function initialize the resolution values with nWidth, nHeight and nDepthNbBits and gets the number of the tree nodes in the octree.

nNumberLimit: This value indicates the limit of the tree nodes to be decoded.

pushQ: This function inserts a value into a queue.

nCount: This value indicates the current number of decoding tree node.

isQueueEmpty: This function checks whether the queue is empty or not.

nIndex: This value indicates the index of the tree node to be decoded.

popQ: This function extracts a value from the queue.

© ISO/IEC 2006 – All rights reserved

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

nSOP: This value indicates whether the tree node is split node or PPM(Prediction by Partial Matching) node. If the value is 0, it means split node. Otherwise, it means ppm node.

decodeAAC: This function performs the AAC(Adaptive Arithmetic Coder) decoding with a given context.

nRegionRed: This value indicates the red color range in a voxel region.

nRegionGreen: This value indicates the green color range in a voxel region.

nRegionBlue: This value indicates the blue color range in a voxel region.

nChild: This value indicates the index of the 8 children nodes decoding the split node.

nBOW: This value indicates whether the child node is black or white.

getCountOfTreeSize: This function calculates the number of sub-tree nodes from a tree node.

getRegion: This function calculates the volume region (starting x, y, z and ending x, y, z) from an index of the tree node.

nStartX, nStartY, nStartZ: These values indicate the starting points of the volume region.

nEndX, nEndY, nEndZ: These values indicate the ending points of the volume region.

nIndexOfContext: This value indicates an index of the tree node context from x, y, z values.

getIndexOfContext: This function gets the index of the tree node context from x, y, z values.

nVoxel: This value indicates whether the voxel node is black or white.

nDeltaRed: This value indicates the differentiated value of the red color in a voxel.

nDeltaGreen: This value indicates the differentiated value of the green color in a voxel.

nDeltaBlue: This value indicates the differentiated value of the blue color in a voxel.

5.6.5 Decoding Process

5.6.5.1 Overview

As shown in Figure 49, there are two parts to decode PointTexture. Those are header decoder and node decoder. The header decoder is to get the resolution information of PointTexture and the percent value how many tree nodes to decode. And the node decoding process is comprised of the following steps:

- Entropy decoding
- Tree node decoding
- Adjustable octree reconstruction
- Voxel data reconstruction

5.6.5.2 Header Decoding

From the nDepthNbBits, the real range of the depth can be obtained as follows.

$$nDepth = 2^{nDepthNbBits + 1}.$$

The resolution of PointTexture is nWidth $\times$ nHeight $\times$ nDepth. From the resolution values, the adjustable octree can be obtained. The adjustable octree has the five labels as follows:

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 22 — Five labels for adjustable octree nodes.

Labels	Comments
S	**Split:** The node is subdivided into 8 nodes
W	**White:** The node consists of all white voxels
B	**Fill black:** The node consists of, or is approximated by, all black voxels
P	**PPM:** The voxel values within the node are encoded by the PPM algorithm
E	**Empty:** The node has no voxel space

To explain the adjustable octree easily, the adjustable quad trees are adopted and used. Figure 50 and Figure 51 show the examples of the adjustable octrees. In the figures, the white voxels are represented by the white boxes and the white circles with W labels. And the non-white voxels are represented by the color boxes and the circles with B labels. If a node has children of S/W/B/P/E labels, the label of the node is S. If a node is to be decoded by PPM(Prediction by Partial Matching) decoding, the label is P. The empty nodes are represented by the dotted boxes and the dotted circles with E labels. Given a resolution, the full octree nodes and the empty nodes can be recognized and be found out. The decoder need not receive any bitstream or information for empty nodes, because it is possible to know the locations of all empty nodes with only the resolution information. In the adjustable octree, the parent node is subdivided into 8 children nodes regularly as equal as possible. Subdividing a parent node, the subdivision order of 8 children nodes is front left-top, front right-top, front left-bottom, front right-bottom, rear left-top, rear right-top, rear left-bottom, and rear right-bottom. In x, y, z axes, the length of each axis is divided into two sub-parts equally if possible. If it is not possible to divide equally, the length of one sub-part is one voxel longer than the length of the other sub-part. For example, Figure 50 shows that it is divided unequally in x-axis as 2 column voxels and 1 column voxel, but it is divided equally in y-axis as 2 row voxels and 2 row voxels.

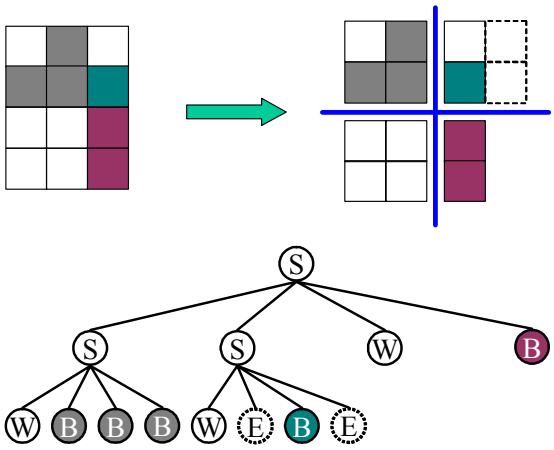


Figure 50 — Example of adjustable octree in the resolution of 3×4.

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

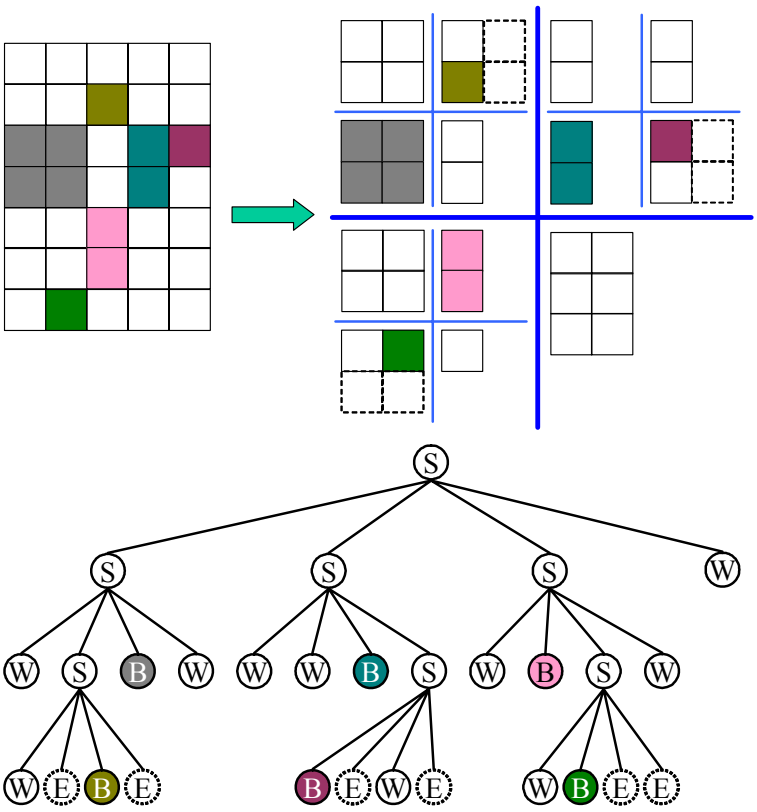


Figure 51 — Example of adjustable octree in the resolution of 5×7.

5.6.5.3 Entropy Decoding

The PointTexture decoder receives the bitstream and decodes the header information and the tree node information. To read the bitstream, the context-based adaptive arithmetic decoder is used as the entropy decoder [42]. The bitstream structure for compressed PointTexture is shown in Figure 52.

The number of the tree nodes (nNumberOfTreeNodes) can be obtained by the initalizeOctree function which make the initial full octree nodes with initializing white (0) values excluding the empty nodes in the resolution of nWidth × nHeight × nDepth. If the number of the tree nodes is N, all nodes to be decoded are Node-1, Node-2, Node-3, …, Node-N.

If the nPercentOfDecoding is 100 (maximum value), all nodes will be decoded losslessly. Otherwise, the tree nodes will be decoded as much as nNumberOfTreeNodes × nPercentOfDecoding / 100.

The header information contains the above resolution values and the percent of decoding. Each information of tree node is composed of two parts as SOP and DIB in Figure 52 (b). SOP is one bit flag, which indicates that the tree node is split node or ppm node. If the node is split node, the bitstream structure is shown in Figure 52 (c). Otherwise, the bitstream structure of ppm node is shown in Figure 52 (d).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2006 – All rights reserved

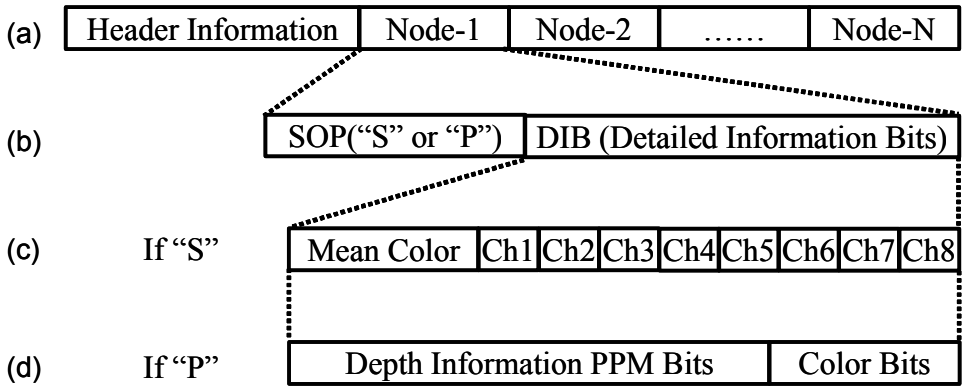


Figure 52 — Example of the bitstream structure for compressed PointTexture.

5.6.5.4 Tree Node Decoding

Given an octree resolution, the number of all tree nodes, N can be calculated and obtained. When the decoder received the bitstream of the tree nodes from root node to leaf nodes, it must know the decoding order in the tree nodes. A modified BFS (Breadth First Search) using a priority order queue can be used for the decoding order. If the decoder use a pure original breadth first search, then the progressive decoding is not possible, but the only sequential decoding is possible. So, modified BFS that is a modified BFS using a priority order queue is proposed and adopted to decode and show the progressive PointTexture. In the children's nodes, every first child node of the parent node is higher than the other child node of the parent nodes. Every second child node is higher than the node from the third to the eighth. Every eighth child node is most low than the other child node of the parent node in the priority. According to the children's priority from a parent node, the current decoding node can be notified to the decoder. Figure 53 shows an example of the decoding order for the tree nodes in Figure 50. In the figure, the empty E nodes are skipped and ignored in the decoding order. Figure 54 shows another example of the decoding order for the tree nodes in Figure 51.

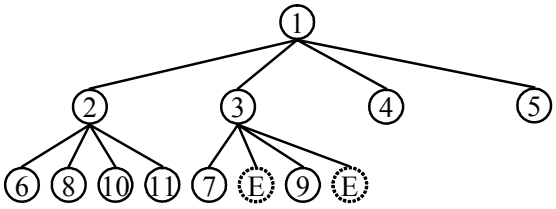


Figure 53 — Example of the decoding order for the tree nodes in Figure 51.

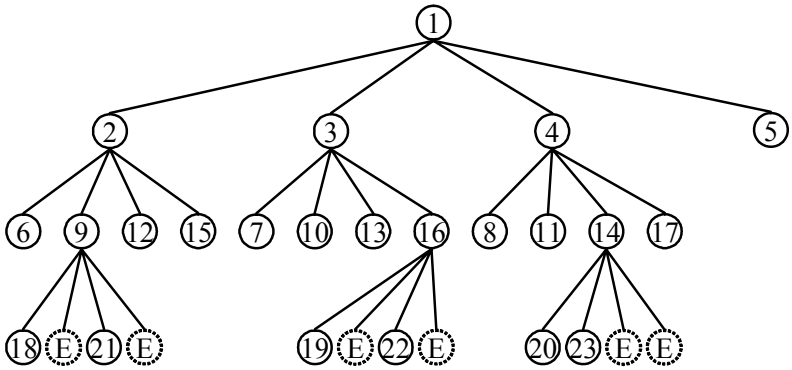


Figure 54 — Example of the decoding order for the tree nodes in Figure 52.

Figure 55 shows the block diagram of the tree node decoder. The decoding procedure is as follows. First, the decoder reads the SOP and finds out whether the tree node is split node or ppm node. Second, the decoder reads the DIB(Detailed Information Bits) and finds out where the voxels are and which colors are in each voxel.

© ISO/IEC 2006 – All rights reserved

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

There are two cases, split node and ppm node. In case of split node, all color values of the children voxels are temporarily set to the average/mean color. These color values are updated when new sub colors of the children are decoded/received. In case of PPM node, the depth information of the voxel region is reconstructed using PPM decoding and the color information is also reconstructed using AAC(Adaptive Arithmetic Coder) decoding and the inverse-DPCM.

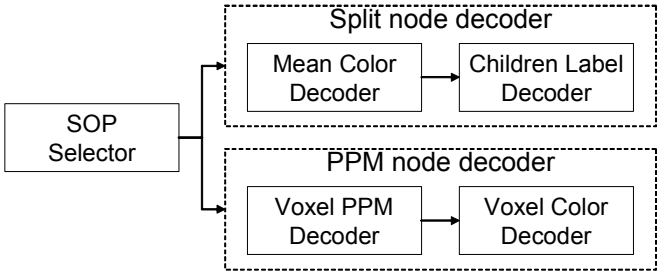


Figure 55 — Block diagram of tree node decoder.

In SOP Selector, the decoder reads the 1 bit flag, which indicates whether the node is the split node or PPM node. If the value is 0, it means split node. Otherwise, it means ppm node. In case of split node, the node is decoded as shown in Figure 52(c). Firstly, the average/mean R, G, B color values are decoded with the adaptive arithmetic decoder. Next, the children labels are also decoded with the adaptive arithmetic decoder. The child label can be black (1) or white (0). If the label is white, all children nodes of the node are white nodes. If the label is black, then the sub children nodes of the label are temporarily regarded as all black nodes. The sub children's nodes and colors can be known in detail when the next sub node of each label is decoded/reached.

In case of PPM node, the node is PPM decoded using the previous decoded voxel values as context. Figure 56 is an example of context. Voxels are represented as circles or squares. Gray circles means decoded black voxels and white circles means decoded white voxels. Question marked circles are not decoded voxels. A question marked black square shown in Figure 56 (b) is the voxel to be decoded. To decode the voxel as 0 or 1, ten neighboring voxels are used as context excluding three voxels which are marked with 'X'. Similar to the raster scan order, '0' is the previous decoded white circled voxel and '1' is the previous decoded gray circled voxel. The ten bits are used as the context of the squared voxels. In this example, the context of the squared voxels is '0111000011'. Using this context, the black squared voxel is decoded by the context-based adaptive arithmetic decoder.

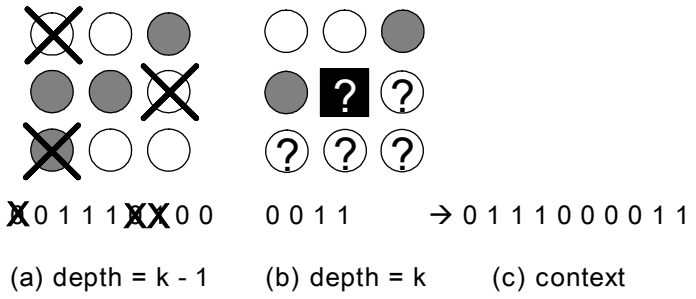


Figure 56 — Example of context: a voxel represented by a question marked black square in (b) is the voxel to be decoded and voxels represented by white circles and gray circles in (a) and (b) are used to make a context excluding three voxels which are marked with 'X'.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

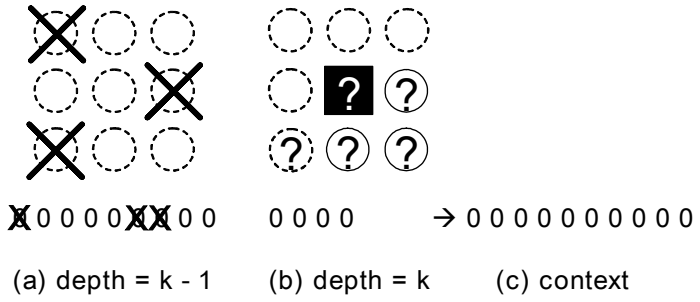


Figure 57 — Example of context: voxels represented by dotted circles in (a) and (b) are not existed voxels and a voxel to be decoded in (b) is located at a corner actually.

After decoding the voxel/depth information, the R, G, B color values of the black voxels within the node are decoded with the adaptive arithmetic decoder and the inverse-DPCM. The inverse-DPCM of color information is performed using the following equations.

nRed = (255 - nDeltaRed) + nRedPrevious , if nDeltaRed > 255

nRed = nDeltaRed + nRedPrevious , if nDeltaRed ≤ 255

nGreen = (255 - nDeltaGreen) + nGreenPrevious , if nDeltaGreen > 255

nGreen = nDeltaGreen + nGreenPrevious , if nDeltaGreen ≤ 255

nBlue = (255 - nDeltaBlue) + nBluePrevious , if nDeltaBlue > 255

nBlue = nDeltaBlue + nBluePrevious , if nDeltaBlue ≤ 255

The values of nDeltaRed, nDeltaGreen, nDeltaBlue are the decoded values after reading and decoding the bitstream of color bits. The values of nRedPrevious, nGreenPrevious, nBluePrevious are the previous decoded values of nRed, nGreen, nBlue. The values of nRed, nGreen, nBlue are the final decoded values which are supposed to be had.

5.6.5.5 Adjustable Octree Reconstruction

In the tree node decoding, the labeled adjustable octree reconstruction is also performed. Figure 58 shows a simple example of the adjustable octree reconstruction in the resolution of 3×4. Figure 58 (b)(d)(f) show the reconstruction process in the decoder side. On the other hand, Figure 58 (a)(c)(e) shows the construction process in the encoder side. In Figure 58 (b)(d)(f), blue line box means the current decoding node and green line box means the decoding children nodes. In Figure 58 (b), the decoding node is split node and the decoding children nodes are B, B, W and B. In Figure 58 (d), the decoding node is PPM node and the decoding children nodes are W, B, B and B. In Figure 58 (f), the decoding node is split node and the decoding children nodes are W and B. In this case, E nodes are not decoded. They can only be recognized and found by the resolution information.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

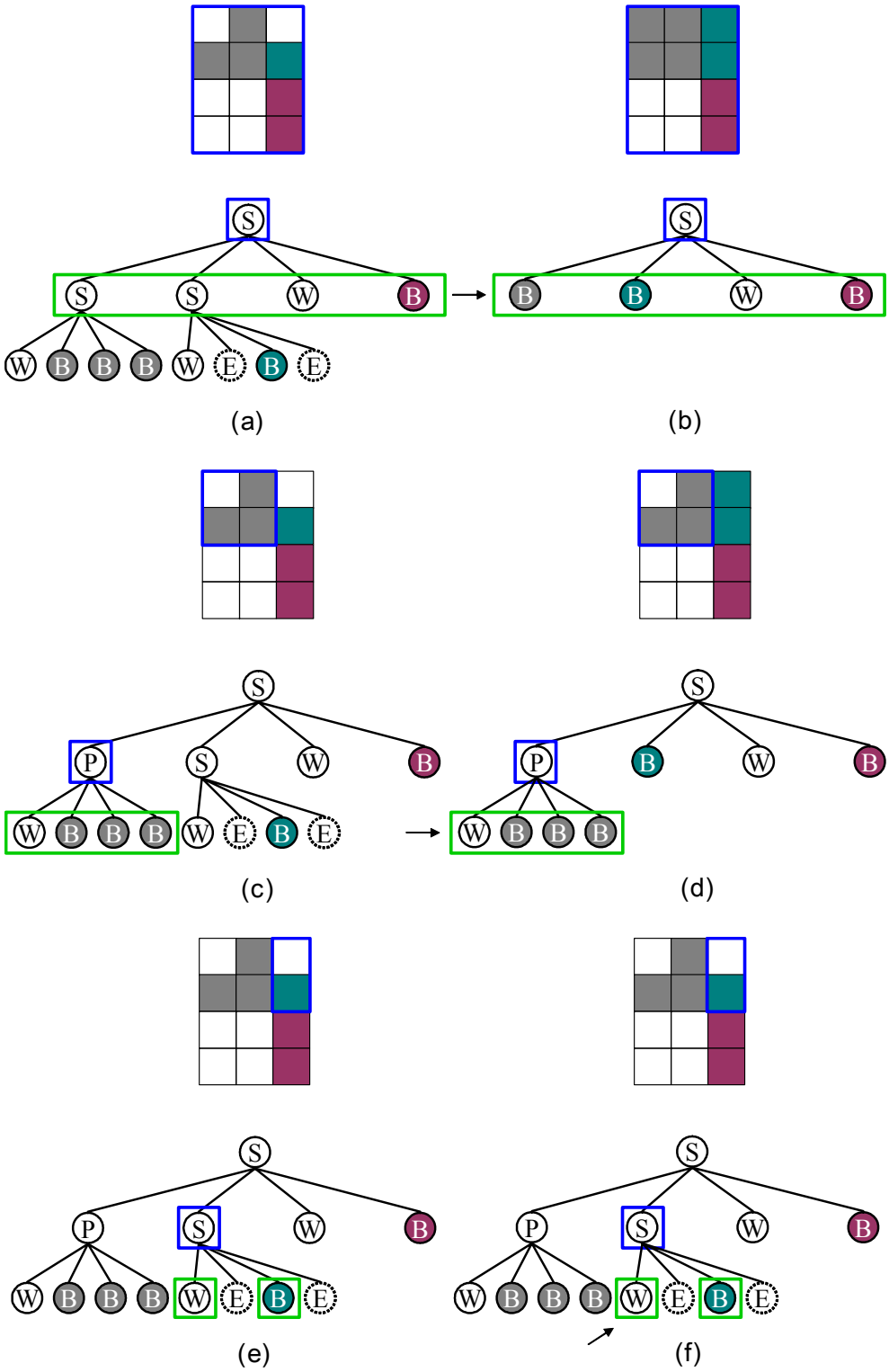


Figure 58 — Example of the adjustable octree construction/reconstruction.

5.6.5.6 Voxel Data Reconstruction

After reconstruction of the labeled adjustable octree, it is converted into the voxel data. The resolution of the voxel data is nWidth × nHeight × nDepth. After reconstruction of the voxel data, it can be converted into the PointTexture easily. The PointTexture node has the depth information and the color information to represent the reconstructed 3D objects. With the labeled octree and the efficient bitstream structure, the progressive decoding is possible.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

5.7 Multiplexing of 3D Compression Streams: the MPEG-4 3D Graphics stream (.m3d) syntax

When coded 3D compression objects are carried without MPEG-4 System, the 3D object elementary streams follow the syntax below. The syntax provides for the multiplexing of multiple elementary streams into a single bitstream.

The *3DCObjectSequence* defines container that is used to carry the *3DCObjectSequence* header and the *3DCObject*. The *3DCObjectSequenceHeader* defines the identification of profile and level for this bitstream and the user data defined by users for their specific applications. For example, it can contain scene information for the contained bitstream.

The *3DCObject* defines container that is used to carry the *3DCObject* header and the 3D compressed bitstream: 3D Mesh Compression (3DMC), Interpolator Compression (IC), Wavelet Subdivision Surface (WSS) and Bone-based Animation (BBA). In the *3DCObject* header, the user data also can be defined by users for their specific applications for 3D compression object. The *3DCObject* header contains the "*3dc_object_verid*" which indicates the version number for the tool list of 3D compression object types. And, "*3dc_object_type_start_code*" indicates what 3D compression object type stream is carried and its correspondence decoder. (i.e. "*3dc_object_type_start_code == Simple_3DMC*", the 3DMC decoder decodes the contained bitstream). For other types of 3D compression objects, one container per 3D compression object is used.

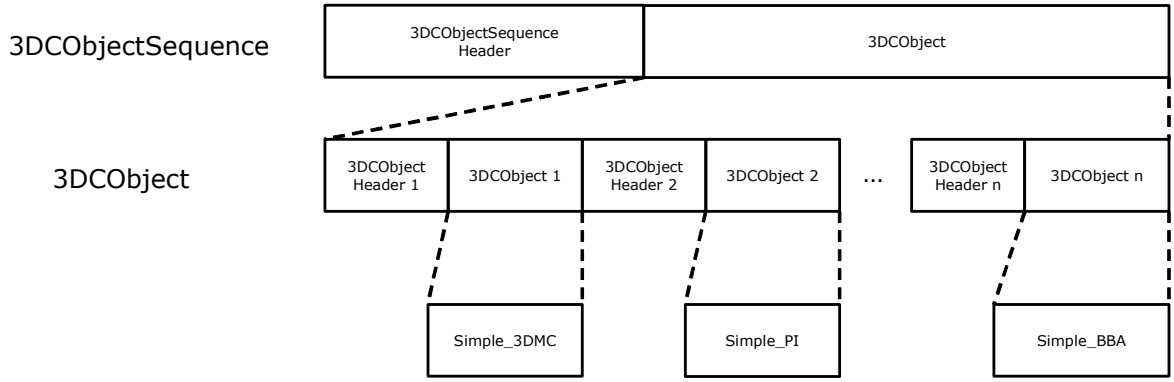


Figure 59 — Syntax of the MP43D stream

5.7.1 3DCObjectSequence

5.7.1.1 Synthax

```
class 3DCObjectSequence (){
   bit(32) 3dc_object_sequence_start_code;
   bit(8)    profile_and_level_indication;
   bit(32) *next;
   while (next == user_start_code) {
user_data();
}
   do {
      3DCObject();
      bit(32) *next;
   }while(next != 3dc_object_sequence_end_code );
   bit(32) 3dc_object_sequence_end_code;
}
```

© ISO/IEC 2006 – All rights reserved

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

5.7.1.2 Semantics

3dc_object_sequence_start_code: The 3dc_object_sequence_start_code is the bit string '000001A0' in hexadecimal. It initiates a 3D Compression 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 23.

Table 23 — FLC table for profile_and_level_indication.

Profile/Level	Code
Reserved	00000000
Core Profile/Level 1	00000001
Core Profile/Level 2	00000010
Reserved	00000011
:	:
Reserved	11111111

3dc_object_sequence_end_code: The 3dc_object_sequence_end_code is the bit string '000001A1' in hexadecimal. It terminates a 3D Compression session.

5.7.2 3DCObject

5.7.2.1 Synthax

```
class 3DCObject(){
   bit(32) 3dc_object_start_code ;
   bit(1)  is_3dc_object_identifier;
   bit(3)  3dc_object_verid;
   bit(4)  3dc_object_priority;

   bit(32) *next;
   while (next == user_start_code) {
    user_data();
   }
   bit(32) *next;
   if (next == "Simple_3DMC") {
      bit(32) simple_3dc_object_type_start_code;
      3D_Mesh_Object() ;
   }
   else if (next == "Simple_WSS") {
      bit(32) simple_WSS_object_type_start_code;
      Wavelet_Mesh_Object();
   }
   else if (next == "Simple_CI") {
      bit(32) simple_CI_object_type_start_code;
      CompressedCoordinateInterpolator() ;
   }
   else if (next == "Simple_OI") {
      bit(32) simple_OI_object_type_start_code;
      CompressedOrientationInterpolator () ;
   }
   else if (next == "Simple_PI") {
      bit(32) simple_PI_object_type_start_code;
      CompressedPositionInterpolator ();
   }
```


© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   else if (next == "Simple_BBA") {
      bit(32) simple_BBA_type_start_code;
      bba_object ();
   }
}
```

5.7.2.2 Semantics

3dc_object_start_code: The 3dc_object_start_code is the bit string '000001A2' in hexadecimal. It initiates a 3D Compression object.

is_3dc_object_identifier: This is a 1-bit code which set to '1' indicates that version identification and priority is specified for the 3D Compression object.

3dc_object_verid: This is a 4-bit code which identifies the version number of the 3D Compression object. Its meaning is defined in Table 24. When this field does not exist, the value of 3dc_object_verid is '0001'.

Table 24 — Meaning of 3dc_object_verid.

3dc_object_verid	Meaning
0000	reserved
0001	Object Types listed in Table 25
0010-1111	reserved

Table 25 list the tools included in each of the Object Types. The current object types can be extended when new tools or functionalities will be introduced.

Table 25 — Tools for 3D Compression Object Types.

AFX Tools	3D Compression Object Types					
	Simple 3DMC	Simple CI	Simple PI	Simple OI	Simple WSS	Simple BBA
3D Mesh Compression (3DMC) - Basic	X					
CoordinateInterpolator (CI)		X				
PositionInterpolator (PI) - Key Preserving - Path Preserving			X			
OrientationInterpolator (OI) - Key Preserving - Path Preserving				X		
Wavelet Subdivision Surface (WSS) - IndexedFaceSet for base mesh					X	
BBA - Only Bones						X

3dc_object_priority: This is a 3-bit code which specifies the priority of the 3D compression 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.

© ISO/IEC 2006 – All rights reserved

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

3dc_object_type_start_code: It is the bit string of 32 bits. The first 24 bits are '0000 0000 0000 0000 0000 0001' in binary for resynchronization. The last 8 bits represent the one of the values in the 'A6' to 'AB' to indicated object types defined in Table 26. According to the last 8 bits in **"3dc_object_type_start_code"**, the corresponding decoder is called and the compressed stream is decoded. If more object types are defined in Table 25, the added object types are reflected in Table 26.

Table 26 — Meaning of start code value.

3dc_object_type_start_code	code (hexadecimal)
Reserved	A5
Simple 3DMC	A6
Simple CI	A7
Simple PI	A8
Simple OI	A9
Simple WSS	AA
Simple BBA	AB
Reserved	AC through FF

5.7.3 user_data

5.7.3.1 Synthax

```
class user_data()
{
   bit(23) * next;
   while (next != 0) {
      bit(8)  user_data_bits;
      bit(23) * next;
   }
}
```

5.7.3.2 Semantics

user_data_start_code: The user_data_start_code is the bit string '000001A4' 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 AFX Object code

AFXExtDescriptor described in ISO/IEC 14496-1 is an abstract class used as a placeholder for an optional DecoderSpecificInfo defined in Table 27.

Table 27 — DecoderSpecificInfo for AFX streams.

AFX stream	DecoderSpecificInfo
MeshGrid	See subclause 5.2.2.2.
WaveletSubdivisionSurface	See subclause 5.1.1.1.
Other AFX streams	None

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The tag field in the AFXExtDescriptor refers to a specific node compression scheme as defined in Table 28.

Table 28 — AFX object code.

AFX object code	Object
0x00	3D Mesh Compression
0x01	WaveletSubdivisionSurface
0x02	MeshGrid
0x03	CoordinateInterpolator
0x04	OrientationInterpolator
0x05	PositionInterpolator
0x06	OctreeImage
0x07	BBA
0x08	PointTexture

7 3D Graphics Profiles

7.1 Introduction

A 3D graphics profile defines the set of tools that a product or application compliant with that profile must implement. In MPEG-4 there are defined several profile dimensions. The ones of interest for 3D graphics are "Scene Graph", "Graphics" and "3D Compression" dimensions. The first two refers to nodes in the scene graph and the last refers to compression tools. A profile is defined inside a dimension. One product or application can be compliant with only one profile in a dimension but it can combine several profiles belonging to different dimensions.

7.2 "Graphics" Dimension

7.2.1 MPEG-4 X3D Interactive Graphics Profiles and Levels

7.2.1.1 List of tools/functionalities

The X3D Interactive Graphics profile represents a collection of nodes to allow implementation of a low-footprint engine (e.g. a Java applet or small browser plugin) and is intended to address limitations of software renderers. This set of nodes matches with the nodes related to graphics within the set of nodes being used as an Interactive profile within the X3D standard's development.

The following graphics nodes are supported within this profile: Appearance, Box, Color, Cone, Coordinate, Cylinder, ElevationGrid, IndexedFaceSet, IndexedLineSet, Material, PointSet, Shape, Sphere and TextureCoordinate.

7.2.1.2 Comparison with existing profiles

The X3D Interactive Graphics profile is based on X3D level 1 Interactive Profile [98], and adds the following MPEG-4 features: BIFS-commands for streaming and Quantization for compression efficiency. No other profile in MPEG-4 addresses 3D (only) environments.

7.2.1.3 X3D Interactive Graphics Profile @ Level 1 Definition

In the following table, definitions for level 1 of the X3D Interactive Graphics profile are given.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 29 — Level 1 of Web3D Interactive Graphics profile.

Node	Minimum System Support
Appearance	*textureTransform* not supported.
Background	*groundAngle* and *groundColor* not supported. *texture* treated as background image FALSE. *stretchToFit* not supported. *backURL*, *frontURL*, *leftURL*, *rightURL*, *topURL* are not supported. One *skyColor*.
Box	Full support
Color	15,000 colors.
Cone	Full support
Coordinate	65,535 points.
Cylinder	Full support
Directional Light	*AmbientIntensity* not supported. Not scoped by parent Group or Transform.
IndexedFaceSet	*set_colorIndex* not supported. *set_normalIndex* not supported. *ccw* not supported. *normal* not supported. Only convex indexed face sets supported. Hence, *convex* is not supported. For *creaseAngle*, only 0 and pi radians supported. *normalIndex* not supported. 10 vertices per face. 5000 faces. 65,535 indices in any index field. Face list shall be well-defined as follows: 1. Each face is terminated with -1, including the last face in the array. 2. Each face contains at least three non-coincident vertices. 3. A given *coordIndex* is not repeated in a face. 4. The vertices of a face shall define a planar polygon. 5. The vertices of a face shall not define a self-intersecting polygon.
IndexedLineSet	*set_colorIndex* not supported. *set_coordIndex* not supported. *ccw* not supported. 15,000 total vertices. 15,000 indices in any index field.
Material	*AmbientIntensity* not supported. *shininess* not supported. *SpecularColour* not supported. A Material with emissiveColour not equal to (0,0,0), diffuseColor equal to (0,0,0) is an unlit Material. One-bit transparency; transparency values >= 0.5 transparent.
PointLight	Ignore *radius*. Ignore *ambientIntensity*. Linear attenuation.
PointSet	5000 points.
Shape	Full support.
Sphere	Full support
SpotLight	Ignore *beamWidth*. Ignore *radius*. Ignore *ambientIntensity*. Linear attenuation.
TextureCoordinate	65,535 coordinates.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 30 specifies further restriction to the fields of the nodes listed in Table 29. These tables can be used for both the Profile and the Level definitions.

Table 30 — Functionality limitation and minimum system requirement.

Node	Restrictions
All lights	8 simultaneous lights.
Names for DEF/PROTO/field	50 utf8 octets.
All *url* fields	10 URLs.
SFBool	Full support.
SFColor	Full support.
SFFloat	Full support.
SFImage	256 width. 256 height.
SFInt32	Full support.
SFNode	Full support.
SFRotation	Full support.
SFString	30,000 utf8 octets.
SFTime	Full support.
SFVec2d	15,000 values.
SFVec2f	15,000 values.
SFVec3d	15,000 values.
SFVec3f	15,000 values.
MFColor	15,000 values.
MFFloat	1,000 values.
MFInt32	20,000 values.
MFNode	500 values.
MFRotation	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.
MFVec2d	15,000 values.
MFVec2f	15,000 values.
MFVec3d	15,000 values.
MFVec3f	15,000 values.

Note: The X3D interactive Profile is a common compatibility point with We3D's X3D Interactive Profile. The following tools are supported in MPEG, but not in Web3D's profile: *QuantizationParameter*, *Node Update*, *Scene Update* and *Route Update*.

7.3 "Scene Graph" Dimension

7.3.1 MPEG-4 X3D Interactive Scene Graph Profile and Levels

7.3.1.1 List of tools/functionalities

The X3D Interactive Scene Graph profile represents a collection of nodes to allow implementation of a low-footprint engine (e.g. a Java applet or small browser plugin) and is intended to address limitations of software renderers. This set of nodes matches with the nodes related to scene description within the set of nodes being used as a Interactive profile within the X3D standard's development.

The following scene graph nodes are supported in this profile: Background, DirectionalLight, PointLight and SpotLight.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.3.1.2 BIFS nodes for support of Audio and Visual objects

If this profile is used in combination with an Audio and/or a Visual Profile, the required nodes are inferred from these Audio/Video Profiles respectively.

If this profile is used in combination with an Audio Profile: Sound, AudioClip,

If this profile is used in combination with a Visual Profile ImageTexture MovieTexture.

7.3.1.3 Comparison with existing profile

No other profile in MPEG-4 addresses 3D-only interactive environments. The existing 'Complete' profile is a much larger set of tools.

7.3.1.4 MPEG-4 X3D Interactive Scene Graph Profile @ Level 1 Definition

In the following table, definitions for level 1 of the X3D Interactive Scene Graph profile are given.

Table 31 — Level 1 of X3D Interactive Scene Graph profile.

Node	Minimum Browser Support
Anchor	*addChildren* not supported. *removeChildren* not supported. Ignore *parameter*.
ColorInterpolator	Full support.
CoordinateInterpolator	15,000 coordinates per *keyValue*.
CylinderSensor	Full support.
Group	*AddChildren* not supported. *removeChildren* not supported.
Inline	Full support.
NavigationInfo	*avatarSize* not supported. *speed* not supported. Ignore *visibilityLimit*
OrientationInterpolator	Full support.
PlaneSensor	Full support.
PositionInterpolator	Full support.
ProximitySensor	Full support.
ScalarInterpolator	Full support.
SphereSensor	Full support.
Switch	Full support.
TimeSensor	Ignored if *cycleInterval* < 0.01 second.
TouchSensor	*hitNormal_changed* not supported
Transform	*addChildren* not supported. *RemoveChildren* not supported.
Viewpoint	Ignore *fieldOfView*.
WorldInfo	Full support
QuantizationParameter	Full support
Node Updates	Full support
Scene Updates	Full support
Route Updates	Full support

The X3D Interactive Graphics profile is based on X3D level 1 Interactive Profile, and adds the following MPEG-4 features: BIFS-commands for streaming and Quantization for compression efficiency.

--`,,```,,,,````-`-,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2006 – All rights reserved
Not for Resale

Table 32 specifies other aspects of functionality that are supported by this profile. Note that general items refer only to those specific nodes listed in Table 31.

Table 32 — Functionality Limitations and Minimum System Requirements.

Node	Minimum System Support
All groups	500 children. Ignore *bboxCenter* and *bboxSize*.
All interpolators	1000 key-value pairs.
Names for DEF/PROTO/field	50 utf8 octets.
All *url* fields	10 URLs
SFBool	Full support.
SFFloat	Full support.
SFImage	256 width. 256 height.
SFInt32	Full support.
SFNode	Full support.
SFRotation	Full support.
SFString	30,000 utf8 octets.
SFTime	Full support.
SFVec2d	15,000 values.
SFVec2f	15,000 values.
SFVec3d	15,000 values.
SFVec3f	15,000 values.
MFFloat	1,000 values.
MFInt32	20,000 values.
MFNode	500 values.
MFRotation	1,000 values.
MFString	30,000 utf8 octets per string, 10 strings.
MFVec2d	15,000 values.
MFVec2f	15,000 values.
MFVec3d	15,000 values.
MFVec3f	15,000 values.

© ISO/IEC 2006 – All rights reserved

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

7.4 "3D Compression" Dimension

The 3D Compression Dimension is defined on the same level as OD, Video, Audio, Graphics, Scene Graph, MPEGJ and Text dimensions and is signalized as indicated in ISO/IEC 14496-1:2005/Amd.1.

3DCompressionProfileLevelIndication is defined in Table 33.

Table 33 — 3DCompressionProfileLevelIndication Values.

Value	Profile	Level
0x00	Reserved for ISO use	-
0x01	Core	L1
0x02	Core	L2
0x0A-0x7F	reserved for ISO use	-
0x80-0xFD	user private	-
0xFE	no 3D Compression profile specified	-
0xFF	no 3D Compression capability required	-
Note: Usage of the value 0xFE may indicate that the content described by this descriptor does not comply to any conformance point specified in this international standard.		

7.4.1 "Core 3D Compression" Profile and Levels

The "Core 3D Compression" profile combines the 3D Mesh Compression, Interpolation Compression, Wavelet Subdivision Surface, and Bone Based Animation tools for efficient 3D resource transmission and storage.

7.4.1.1 List of Tools/Functionalities

The "Core 3D Compression" profile represents a collection of compression tools to allow implementation of minimum functionalities for compact transmission and storage of 3D object under a constrained environment (e.g. mobile), where the processing power and memory size can be very limited.

The "Core 3D Compression" profile contains the following 3D Compression object types:

- The **Simple 3DMC** object type provides high compression and error resilience for static triangle 3D models.
- The **Simple CI** object type compresses the Coordinate Interpolator animation.
- The **Simple PI** object type compresses the Position Interpolator animation. It can support both Key-Preserving and Path-Preserving mode.
- The **Simple OI** object type compresses the Orientation Interpolator animation. It can support both Key-Preserving and Path-Preserving mode.
- The **Simple WSS** object type represents, in a compressed form, the details for subdivision of 3D mesh. This tool is used for level of detail management and animation. It only allows IndexedFaceSet as a base mesh.
- The **Simple BBA** object type compresses the skeleton animation based on bone transforms and connected to a skin mesh model. This object type does not support Muscle.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 34 — 3D Compression Object Types.

AFX Tools	3D Compression Object Types					
	Simple 3DMC	Simple CI	Simple PI	Simple OI	Simple WSS	Simple BBA
3D Mesh Compression (3DMC) - Basic	X					
CoordinateInterpolator (CI)		X				
PositionInterpolator (PI) - Key Preserving - Path Preserving			X			
OrientationInterpolator (OI) - Key Preserving - Path Preserving				X		
Wavelet Subdivision Surface (WSS) - IndexedFaceSet for base mesh					X	
BBA - Only Bones						X

The "Core 3D Compression" includes the object types as illustrated in Table 34.

Table 35 — "Core 3D Compression" Profile.

	"3D Compression" Object Types					
	Simple 3DMC	Simple CI	Simple PI	Simple OI	Simple WSS	Simple BBA
Core 3D Compression Profile	X	X	X	X	X	X

7.4.1.2 Comparison with Existing Profiles and object types

The Core 3D Compression Profile is the first profile defined in the "3D Compression" profile dimension.

7.4.1.3 Profile Level Definition

According to target device and applications, we defined two levels as listed in Table 36, Table 37and Table 38. The level 1 is for the mobile device without H/W graphics accelerator. Thus, it is suitable simple application such as 3D Background, 3D GUI, 3D Pre-Viewer and 3D Avatar. On the other hand, the level 2 is for the mobile device supported by H/W graphics accelerator. Thus, the level 2 is suitable for the 3D Game application in addition to the application in level 1.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 36 — Levels and data constraints.

Level	Data Constraints
Level 1	- Number of triangles in a scene ≤ 500 - Number of objects in a scene ≤ 30 - Number of bones attachable to an object ≤ 10 - Number of bones affecting a vertex ≤ 2 - Number of vertex in Coordinate Interpolator = 0
Level 2	- Number of triangles in a scene ≤ 5000 - Number of objects in a scene ≤ 100 - Number of bones attachable to an object ≤ 30 - Number of bones affecting a vertex ≤ 4 - Number of vertex in Coordinate Interpolator > 0

Table 37 — Levels and functionalities constraints.

Level	Functionality Constraints
Level 1	- Triangle only - colorIndex and texCoordIndex not supported (coordIndex values are used instead) - The vertices of a face shall not define a self-intersecting polygon - A given coordIndex is not repeated in a face - Ignore normal and normalIndex - The color and texCoord cannot be supported at the same time. - Only base mesh supported
Level 2	- Triangle only - Full support of colorIndex, normalIndex, texcoordIndex - The vertices of a face shall not define a self-intersecting polygon - A given coordIndex is not repeated in a face - Color and Texture can be blended. - Only base mesh supported

Table 38 — Levels and player constraints.

Level	Player Constraints (Informative Recommendation)
Level 1	- Size of Texture memory ≤ 1 M - Number of light ≤ 1 - DirectionalLight - Material: ambientIntensity, diffuseColor, emissiveColor - Transparency - Flat and Gouraud shading - Texture Blending not supported
Level 2	- Size of Texture memory ≤ 4 M - Number of light ≤ 8 - Directional Light, PointLight and SpotLight - Material: ambientIntensity, diffuseColor, emissiveColor, shininess, specularColor - Transparency - Flat and Gouraud shading - Texture Blending - Texture Filtration - Perspective correction - Total Decoding Time ≤ 10sec

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 XMT representation for AFX tools

8.1 AFX nodes

The XMT syntax for all the nodes defined in this international standard is specified in the xsd file attached to ISO/IEC 14496-11.

8.2 AFX encoding hints

8.2.1 WaveletSubdivision encoding hints

8.2.1.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.2.1.2 Semantics

The <BitWrapperWaveletSubdivisionEncodingHints> allow the source and/or target formats to be described so that when encoding is required the author can control various parameters of the encoding process where supported by the encoder.

8.2.2 MeshGrid encoding hints

8.2.2.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.2.2.2 Semantics

The <BitWrapperMeshGridEncodingHints> allow the source and/or target formats to be described so that when encoding is required the author can control various parameters of the encoding process where supported by the encoder.

8.2.3 OctreeImage encoding hints

8.2.3.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.2.3.2 Semantics

The <BitWrapperOctreeImageEncodingHints> allow the source and/or target formats to be described so that when encoding is required the author can control various parameters of the encoding process where supported by the encoder.

8.2.4 PointTexture encoding hints

8.2.4.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.4.2 Semantics

The <BitWrapperPointTextureEncodingHints> allow the source and/or target formats to be described so that when encoding is required the author can control various parameters of the encoding process where supported by the encoder.

8.2.5 BBA encoding hints

8.2.5.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.2.5.2 Semantics

The <BBAEncodingHints> allow the source and/or target formats to be described so that when encoding is required the author can control various parameters of the encoding process where supported by the encoder.

8.3 AFX encoding parameters

8.3.1 WaveletSubdivisionEncodingParameters

8.3.1.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.3.1.2 Semantics

NbBpSC: stores the number of bits on which to encode the scaling coefficients minus one.

NbBPX: stores the number of bits on which to encode the first component of the waelet coefficients.

NbBPY: stores the number of bits on which to encode the second component of the waelet coefficients.

NbBPZ: stores the number of bits on which to encode the third component of the waelet coefficients.

Wtype: stores the type of subdivision scheme used in the low pass filtering.

lift: indicates whether the high pass filtering is lifted or not. This is an integer type.

isInLocalCoordinates: indicates whether the wavelet coefficients are expressed in local or global coordinates.

LengthNbBits: indicates the number of bits used to represent the number of bits used to encode the length of the bitplanes. This is an integer type.

isPartial: indicates whether the transmission is in full mode or bitplane by bitplane. This is an integer type.

8.3.2 MeshGridEncodingParameters

8.3.2.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2.2 Semantics

nLevels: specifies the maximum number of resolution levels (in the {u,v,w} directions of the reference-grid) of the encoded model. If no value is specified, the encoded model will preserver the number of resolution levels of the original MeshGrid model.

nSlices: specifies the maximum number of slices (in the {u,v,w} directions of the reference-grid) corresponding to the last resolution level of the encoded model. If no value is specified, the encoded model will preserver the number of slices of the original MeshGrid model.

sizeROI: specifies the optimal size of the region of interest (ROI). It is used to split the original model into ROIs when encoding. If no value is specified for **sizeROI**, the model is encoded at each resolution level as one entity. The number of ROIs at the highest resolution level of the model is computes as explained in subclause C.3.

hasConnectivityInfo: specifies if true that the connectivity should be encoded. If no value is specified, the value from the original mesh is considered.

hasRefineInfo: specifies if true that the vertices' offsets should be encoded. If no value is specified, the value from the original mesh is considered.

hasRepositionInfo: specifies if true that the vertices' inter-level refinement information should be encoded. If no value is specified, the value from the original mesh is considered.

hasGridInfo: specifies if true that the reference-grid coordinates should be encoded. If no value is specified, the value from the original mesh is considered.

meshType: suggests a particular type of mesh, in case the mesh can be adapted. If no value is specified, the value from the original mesh is considered.

sameBorderOrientation: suggests to encode (if possible) the particular mesh in a more compact way. If no value is specified, the value from the original mesh is considered.

uniformSplit: suggests to encode the particular mesh only at the first resolution level (only if the **hasConnectivityInfo** flag is set as well). If no value is specified, the value from the original mesh is considered.

offsetAmplitude: specifies the amplitude of the vertices' offsets when the model is defined on a single-layer reference-grid. If no value is specified, the value from the original mesh is considered.

cyclicMode: suggests the cyclic mode of the mesh, in case the mesh can be adapted. If no value is specified, the value from the original mesh is considered.

fullRefine: specifies if "true" that the vertices' offsets should be encoded at each resolution level.

nRefineBits: specifies the number of bits to quantize the vertices' offsets.

filterType: specifies the type of filter used for the wavelet transform, and may yield one of the following two values: "SHORT_FILTER" and "SMOOTH_FILTER". If no value is specified, the value from the original mesh is considered.

nQuantBits: specifies the number of bits to quantize the reference-grid coordinates.

gridCR: specifies the compression ratio of the lossy compressed stream compared to the "non-lossy" encoded stream. The compression will affect only the precision of the coordinates of the reference-grid, the connectivity is always encoded lossless.

maxError: specifies the maximum acceptable error between the lossy encoded model and the original model.

© ISO/IEC 2006 – All rights reserved

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

8.3.3 PointTextureEncodingParameters

8.3.3.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.3.3.2 Semantics

codingPercentType : indicates the percent or the level of compression for depth values and color values in the PointTexture. The value 100 indicates lossless coding. Otherwise, it is lossy coding.

8.3.4 BBAEncodingParameters

8.3.4.1 Syntax

The syntax is described in the xsd file attached to 14496-11.

8.3.4.2 Semantics

method: indicates if the encoding method is predictive frame based or DCT-based. The field can have values "PRED" or "DCT"

prediction: indicates the prediction method used for encoding. The field can take one of the values: "Intra", "Predicted", "bestFit" or "PbetweenI". If "Intra", then the current AU must be encoded as an Intra AU. If "Predicted", then the current AU must be encoded as a predictive AU. If "bestFit", then the encoder must take the decision for encoding as an independent AU or a predicted one, while minimizing the amount of data needed for encoding. If "PbetweenI", then the encoding sequence is obtained by one I AU followed by zero or more AUs.

IMax: contains the new values for maximal values used for encoding the Intra AU. This field can be specified only into an Intra AU, and is used until a new IMax is received. If no IMax is specified the values are those defined in 5.4.2.12.2.

IMin: contains the new values for minimal values used for encoding the Intra AU. This field can be specified only into an Intra AU, and is used until a new IMin is received. If no IMin is specified the values are those defined in 5.4.2.12.2.

PMax: contains the new values for maximal values used for encoding the Predictive AU. This field can be specified only into an Intra AU, and is used until a new PMax is received. If no PMax is specified the values are those defined in 5.4.2.12.2.

PMin: contains the new values for maximal values used for encoding the Predictive AU. This field can be specified only into an Intra AU, and is used until a new PMin is received. If no Pmin is specified the values are those defined in 5.4.2.12.2.

Encoding_framerate: indicates the frame rate of the BBA animation.

gqs: indicates an index used for computing the global quantization as defined in 5.4.2.14.2 and 5.4.2.16.2.

p2qs: indicates an order used for computing the global quantization as defined in 5.4.2.14.2 and5.4.2.16.2.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.4 AFX decoder specific info

8.4.1 WaveletSubdivision decoder specific info

8.4.1.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.4.1.2 Semantics

NbBpSC: stores the number of bits on which to encode the scaling coefficients minus one.

NbBPX: stores the number of bits on which to encode the first component of the waelet coefficients.

NbBPY: stores the number of bits on which to encode the second component of the waelet coefficients.

NbBPZ: stores the number of bits on which to encode the third component of the waelet coefficients.

Wtype: stores the type of subdivision scheme used in the low pass filtering.

lift: indicates whether the high pass filtering is lifted or not. This is an integer type.

isInLocalCoordinates: indicates whether the wavelet coefficients are expressed in local or global coordinates.

LengthNbBits: indicates the number of bits used to represent the number of bits used to encode the length of the bitplanes. This is an integer type.

isPartial: indicates whether the transmission is in full mode or bitplane by bitplane. This is an integer type.

8.4.2 MeshGrid decoder specific info

8.4.2.1 Syntax

The syntax is described in the xsd file attached to 14496-11.

8.4.2.2 Semantics

nLevels: specifies the maximum number of resolution levels (in the {u,v,w} directions of the reference-grid) of the encoded model.

nSlices: specifies the maximum number of slices (in the {u,v,w} directions of the reference-grid) corresponding to the last resolution level of the encoded model.

hasConnectivityInfo: specifies if true that the connectivity description is encoded in the bitstream.

hasRefineInfo: specifies if true that the vertices' offsets are encoded in the bitstream.

hasRepositionInfo: specifies if true that the vertices' inter-level refinement information is encoded in the bitstream.

hasGridInfo: specifies if true that the reference-grid coordinates are encoded in the bitstream.

Attributes: reserved.

meshType: specifies the type of the encoded mesh.

sameBorderOrientation: specifies if true that the particular mesh is stored in a more compact way.

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

© ISO/IEC 2006 – All rights reserved

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

uniformSplit: specifies if true that the particular mesh is only encoded at the first resolution level (only if the **hasConnectivityInfo** flag is set as well).

offsetAmplitude: specifies the amplitude of the vertices' offsets when the model is defined on a single-layer reference-grid.

cyclicMode: specifies the cyclic mode of the mesh.

fullRefine: specifies if "true" that the vertices' offsets are encoded at each resolution level.

nRefineBits: specifies the number of bits to quantize the vertices' offsets.

filterType: specifies the type of filter used for the wavelet transform, and may yield one of the following two values: "SHORT_FILTER" and "SMOOTH_FILTER".

gridScale: specifies the scaling factors used when quantizing the grid coordinates.

nQuantBits: specifies the number of bits used to quantize the reference-grid coordinates.

gridCorners: stores the scaled values of the quantized reference-grid corners.

8.5 XMT for Bone-based Animation

XMT for BBA represents the textual format for the animation data when using bones, muscles and morph.

8.5.1 BBA

8.5.1.1 Syntax

The syntax is described in the xsd file attached to 14496-11.

8.5.1.2 Semantics

BBA_header: specifies the header of the BBA animation data. Their components are specified in subclause 8.5.2

BBA_body: specifies the body of the BBA animation data. Their components are specified in subclause 8.5.4

8.5.2 BBA_header

8.5.2.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.5.2.2 Semantics

BBA_encoding: specifies the encoding information for the BBA animation data. Their components are specified in subclause 8.5.3

BBA_version: specifies the version of the BBA animation data.

BBA_name: specifies the name of the BBA animation data.

BBA_frameRate: specifies the framerate of the BBA animation data.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.3 BBA_encoding

8.5.3.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.5.3.2 Semantics

encodingType: specifies how the encoder parameters are specified : if "perAU" then the encoder parameters are individually specified for each AU; if "perFile", then the encoding parameters are the same for all tha data to be encoded.

BBAEncodingParameters: specifies the encoding parameters. This type is defined in subclause 8.3.4.

8.5.4 BBA_body

8.5.4.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.5.4.2 Semantics

BBA_frame: specifies the BBA animation data for one frame. The elements of the BBA_frame are described in subclause 8.5.5

8.5.5 BBA_frame

8.5.5.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.5.5.2 Semantics

BBA_frameMask: specifies the mask of one animation frame. The elements of the BBA_frame are described in subclause 8.5.6

BBA_frameValues: specifies the values of one animation frame. The elements of the BBA_frame are described in subclause 8.5.10

8.5.6 BBA_frameMask

8.5.6.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.5.6.2 Semantics

NIF: specifies the numbers of frame to interpolate between the current frame and the one currently received.

NCT: specifies number of controller types.

NSBB: specifies number of bones to be animated in the frame currently received.

NSBM: specifies number of muscles to be animated in the frame currently received.

NSBMF: specifies number of morphs to be animated in the frame currently received.

© ISO/IEC 2006 – All rights reserved

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

BoneMask: specifies the animation mask for a bone. This element is described in subclause 8.5.7.

MuscleMask: specifies the animation mask for a muscle. This element is described in subclause 8.5.8.

MorphMask: specifies the animation mask for a morph. This element is described in subclause 8.5.9.

8.5.7 BoneMask

8.5.7.1 Syntax

The syntax is described in the xsd file attached to 14496-11.

8.5.7.2 Semantics

IDB: specifies the ID of the current bone

isT: specifies if the current bone is translated.

isTX: specifies if the current bone is translated on X.

isTY: specifies if the current bone is translated on Y.

isTZ: specifies if the current bone is translated on Z.

isR: specifies if the current bone is rotated.

isQ: specifies if the rotation of the current bone is specified by using quaternions.

isRAx1: specifies if the current bone is rotated with respect to the first axis. The rotation order is specified by the rotationOrder field of the SBBone node. This field is used when isQ=0.

isRAx2: specifies if the current bone is rotated with respect to the second axis. The rotation order is specified by the rotationOrder field of the SBBone node. This field is used when isQ=0.

isRAx3: specifies if the current bone is rotated with respect to the third axis. The rotation order is specified by the rotationOrder field of the SBBone node. This field is used when isQ=0.

isRQx: specifies if the X component of the quaternion of the current bone is updated This field is used when isQ=1.

isRQy: specifies if the Y component of the quaternion of the current bone is updated This field is used when isQ=1.

isRQz: specifies if the Z component of the quaternion of the current bone is updated This field is used when isQ=1.

isRQw: specifies if the W component of the quaternion of the current bone is updated This field is used when isQ=1.

isS: specifies if the current bone is scaled.

isSX: specifies if the current bone is scaled on X.

isSY: specifies if the current bone is scaled on Y.

isSZ: specifies if the current bone is scaled on Z.

isSO: specifies if the scale orientation field of the current bone is updated.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

isSOX: specifies if the X component of the scale orientation field of the current bone is updated.

isSOY: specifies if the Y component of the scale orientation field of the current bone is updated.

isSOZ: specifies if the Z component of the scale orientation field of the current bone is updated.

isSOA: specifies if the angle component of the scale orientation field of the current bone is updated.

isC: specifies if the center of the current bone is updated.

isCX: specifies if the center of the current bone is updated on X.

isCY: specifies if the center of the current bone is updated on Y.

isCZ: specifies if the center of the current bone is updated on Z.

8.5.8 MuscleMask

8.5.8.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.5.8.2 Semantics

IDM: specifies the ID of the current muscle.

NCP: specifies the number of the control points that are updated in the current frame for the current muscle.

NK: specifies the number of the knots that are updated in the current frame for the current muscle.

MM: specifies a binary vector indicating which control point and which knot are updated in the current frame, for the current muscle.

8.5.9 MorphMask

8.5.9.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

8.5.9.2 Semantics

IDMF: specifies the ID of the current morph.

NT: specifies the number of weights that are updated in the current frame for the current morph.

MFM: specifies a binary vector indicating which weight is updated in the current frame, for the current morph.

8.5.10 BBA_frameValues

8.5.10.1 Syntax

The syntax is described in the xsd file attached to ISO/IEC 14496-11.

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

© ISO/IEC 2006 – All rights reserved

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

8.5.10.2 Semantics

BoneAnimationValue: specifies the updated values of the current bone, with respect to the BoneMask.

MuscleAnimationValue: specifies the updated values of the current bone, with respect to the MuscleMask.

MorphAnimationValue: specifies the updated values of the current bone, with respect to the MorphMask.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(normative)

Wavelet Mesh Decoding Process

A.1 Overview

The typical Wavelet Mesh decoding process is summarized in the following block diagram :

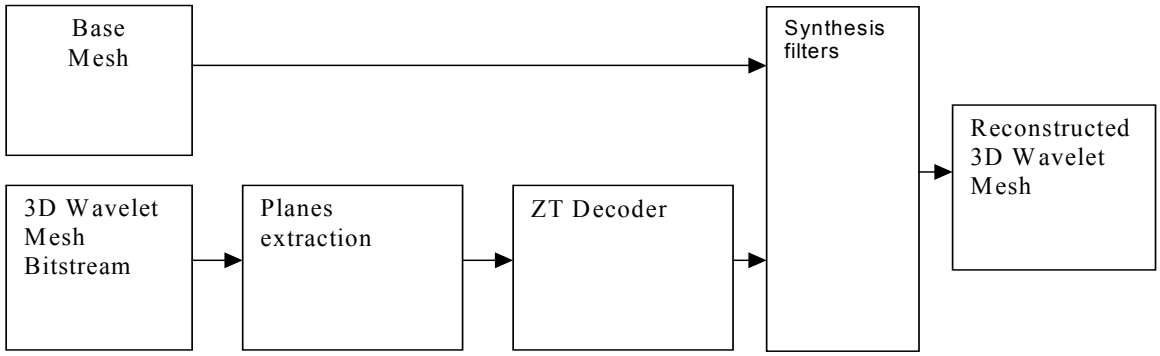


Figure A.1 — The typical Wavelet Mesh decoding process.

This subclause provides a constructive definition of the decoded mesh in function of previous variables read in the bitstream. The way the mesh is rebuilt is not normative.

In the next subclauses, *decoded mesh* will denote a class of meshes having the same geometry. The choice for topology for reconstructing such a mesh is not normative and is left to the decoder.

NOTE – this allows the decoder to adapt to a particular hardware setting. For example if the terminal has a good graphic card but poor CPU power, it may be advantageous to use algorithms producing facets faster, but possibly with a richer topology than the minimum needed.

A.2 Base mesh

The mesh reconstruction can only begin after transmission and decoding of the base mesh.

A.3 Definitions and notations

Let M_0 denote the base mesh, which is supposed to be entirely transmitted before the wavelet coefficients stream starts being decoded.

Let M_j denote the mesh obtained after j iterations of midpoint subdivision of each facet of M_0.

Scaling/wavelet coefficients can be indexed by vertices of the reconstructed mesh, and the level of subdivision at which they occur. Hence, we will denote any coefficient c indexed by level j and vertex v as c^j_v.

The *offspring O(c)* of a coefficient c is the set of coefficients $\{c_0, c_1, c_2, c_3\}$ as in the following figure:

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

© ISO/IEC 2006 – All rights reserved

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

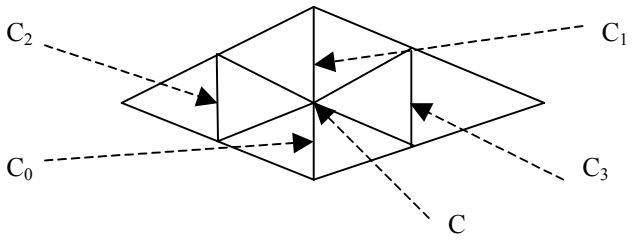


Figure A.2 — Definition of the *offspring*.

The arrows denote the vertices associated to the coefficients, the triangulation figuring the midpoint subdivision of two adjacent facets. For level 1 hierarchy, the order of the children indices is given by the following rule: C_0 and C_1 lie on the same level 0 edge as C; C_2 lies on the first level 0 facet containing C in the base mesh description. This orientation is kept for all the other levels of the same hierarchy. The forest spanned by the level 1 vertices is ordered according to the order of appearance of the corresponding edges in the base mesh facet list.

The *descent D(c)* of a coefficient c is the transitive closure of the relation “is offspring of”.

For any 3-vector or 3-vector-valued function c, let c_x, c_y, and c_z denote respectively the first, second and third component of c.

For any 3-column matrix or 3-column matrix-valued function c, let c_x, c_y, and c_z denote respectively the first, second and third column of c.

A.4 Bitplanes extraction

While receiving the Wavelet Mesh bitstream, the decoder stores in memory the contiguous bits related to a given bitplane in a given zero-tree. Such a storage can be figured as follows, for each table Bx, By, Bz representing the three components of the wavelet coefficients :

First Zero-tree:

Sorting bits + signs	
Sorting bits + signs	Refinement bits
Sorting bits + signs	Refinement bits

…

Sorting bits + signs	Refinement bits

Second Zero-tree:

…

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Last Zero-tree:

Sorting bits + signs	
Sorting bits + signs	Refinement bits
Sorting bits + signs	Refinement bits

…

Sorting bits + signs	Refinement bits

NOTE - the zero-tree decoder is thus completely independent of the bitstream interpretation process and can be run in a separate thread.

This structure corresponds to the entries of table B: B[i][j][k] is stored in the i-th zero-tree space, at line j and column k.

A.5 Zero-tree decoder

Algorithm II of [70] is applied as is with previous definition of *offspring* and *descent*, on each table Bx, By and Bz, using values **NbBPX, NbBPY, NbBPZ** (number of bitplanes for each component) defined in **`WMDecoderConfig`**, and variables explicited above, for recovering respectively three lists Lx, Ly and Lz of wavelet coefficients $2^{-j}(d_{xi}^{j}, d_{yi}^{j}, d_{zi}^{j})_{i,j}$ used in the following subclause for reconstruction.

A.6 Synthesis filters and mesh reconstruction

We describe here the standard, non-algorithmic mesh reconstruction process using the above decoded/stored wavelet coefficients. The scope of adaptivity will thus be restricted to the transmission: no normative directives will constrain the way the client selects the data contained in his cache for partially reconstructing the mesh, according to frame-rate, LoD management etc.

The topology (facets configuration) of the fully reconstructed mesh is semi-regular: it corresponds to the topology of M_j.

The geometry (vertices position in space) of the mesh can be recovered using the decoding process described in [52]. Explicitly, let $(c^0{}_i)_i$ be the sum of the decoded scale coefficients and the corresponding vertices of the base mesh.

Then the positions $c^j{}_i$ of respective vertices v_i at level j can be recovered by iterating the following relations:

$$c_a^{j+1} = T^j \begin{pmatrix} c_a^j \\ f_a(d^j) \end{pmatrix}, \quad a \in \{x, y, z\},$$

over $j=0, \ldots, j_{max}-1$. Function f, which is a basis change transformation from local coordinates to global coordinates, is detailed in the following section. T^j is the synthesis filter matrix: for Loop wavelets, these matrices are detailed in [48]. For butterfly and midpoint subdivision the application of these matrices can be obtained by the following algorithm:

1. Set the current mesh *CM* as the base mesh and set the current level j to 0

© ISO/IEC 2006 – All rights reserved

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

2. For each vertex *i* of *CM*, add $w_i \sum_{s \in S_i} d_s^j$ to vertex *i* position c_i^{j+1}, with

$$w_i = -\mathbf{lift} \cdot \frac{6}{8n_i} \frac{\sin(\pi/3)}{\sin(2\pi/n_i)},$$

n_i being the valence of vertex *i*, and S_i the set of vertices as in Figure A.3.

3. Operate one subdivision step on *CM*, using butterfly (as in the *Subdivision Surfaces* subclause 4.3.3) or midpoint subdivision (i.e. butterfly with w parameter set to 0), according to **`Wtype`**. Vertices created in this process will be said to be *odd.* Every other vertex will be said to be *even*.

4. For each odd vertex *k* in *CM*, add d_k^j to its position c_k^{j+1}.

5. if *j* is different of **`NumberOfLevels`**-1, increment *j* and go to step 2.

In this algorithm, the wavelet coefficients are assumed to be in global coordinates, i.e. after possible application of function *f* in case **`isInLocalCoordinates`** has value 1.

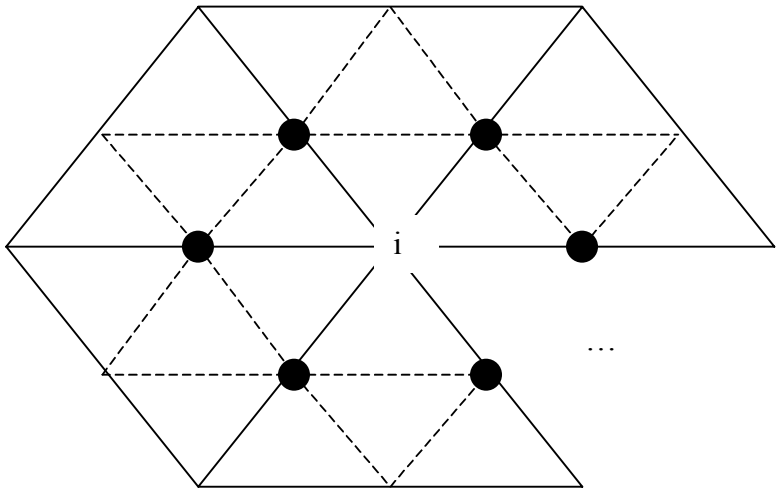


Figure A.3 — The set S_i of vertices of M_{j+1} associated to vertex *i* is figured by the black dots. Continued lines represent triangles in M_j. Dotted lines represent triangles in M_{j+1}.

A.7 Basis change

If **`isInLocalCoordinates`** value is 0, *f* is the identity. Otherwise, for *j*>0, $f(d^j_i)$ is defined as follows:

Let S_j denote the mesh having the same connectivity as M_j but whose vertices have c^j as coordinates : vertex *i* of S_j has c^j_i as coordinates.

Let *A=[b, c]* be the edge of M_j of which vertex *i* in M_{j+1} is the midpoint. Let N_i, U_i, V_i be the vectors defined as follows:

if there exist two different facets in S_j containing *A*, then N_i is the normalized average of the normals to these two facets. Otherwise, it is the normal to the only facet of S_j having *A* as an edge.

$$U_i := \overrightarrow{bc} / \left\| \overrightarrow{bc} \right\|$$

$V_i := N_i \times U_i$

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Then $f(d^j) = \left(f_x(d^j), f_y(d^j), f_z(d^j)\right) := \begin{pmatrix} A_0 {d_0^j}^T \\ \cdots \\ A_i {d_i^j}^T \\ \cdots \end{pmatrix}$,

where A_i is the matrix (N_i U_i V_i) composed with column vectors N_i, U_i and V_i.

© ISO/IEC 2006 – All rights reserved

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

Annex B
(normative)

MeshGrid Representation

B.1 The hierarchical multi-resolution MeshGrid

In the multi-resolution MeshGrid representation both the connectivity-wireframe (see Figure B.1) and the reference-grid (see Figure B.2) have a hierarchical structure. The hierarchical structure enforces that vertices found in a lower level are available in all higher levels that follow. However, each level will alter the connectivity between the existing vertices, e.g. vertex v_n^l is connected on level l with vertex v_m^l via the blue colored line; this connection is replaced in level *l*+1 by another line (green color) to vertex v_p^{l+1}, and replaced again in level *l*+2 by the red line that connects it to vertex v_q^{l+2}. Note that the level of a vertex indicates the position in the hierarchy when it first appears.

The hierarchical MeshGrid model imposes the following constraint on the reference-system: the reference-system of any level is a super-set of the reference-system of the lower levels. Figure B.2 shows the changes in the reference-system when generating a hierarchical MeshGrid with 3 levels: the first level in Figure B.2(a) consists of 3 reference-surfaces colored in blue, the second level in Figure B.2(b) has in addition other reference-surfaces (colored in green), and the third level in Figure B.2(c) contains besides the reference-surfaces of the previous levels also other reference-surfaces colored in red. Each higher level will add an extra reference-surface in between two existing reference-surfaces from the previous levels, while keeping the reference-surfaces of the previous levels.

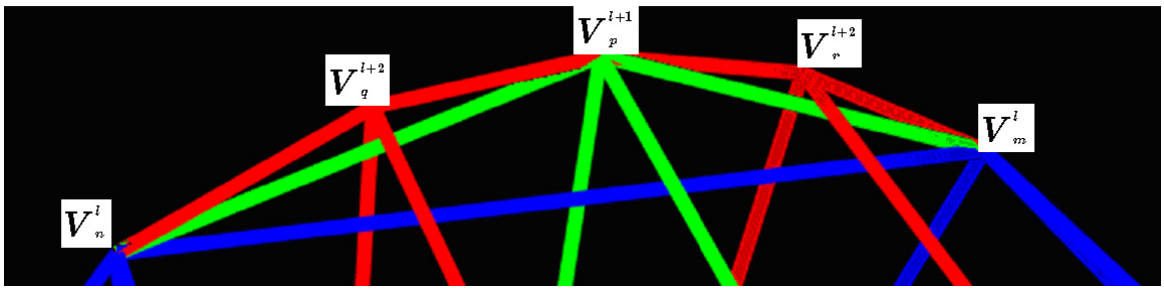
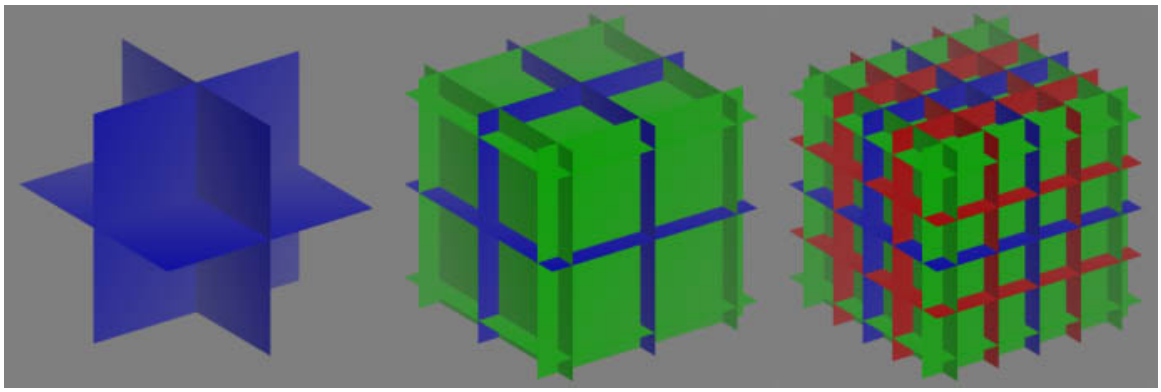


Figure B.1 — The hierarchical connectivity-wireframe of the MeshGrid representation

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



(a) (b) (c)

Figure B.2 — The changes in the reference-system when generating a hierarchical MeshGrid with 3 levels: the first level in (a) consists of 3 reference-surfaces shown in blue color, the second level in (b) has in addition other reference-surfaces that are shown in green color, and the third level in (c) contains besides the reference-surfaces of the previous levels other reference-surfaces shown in red color.

B.1.1 Relation between the resolution level of the mesh and the number of reference-surfaces.

The number of resolution levels *nLevels* is specified for each direction $d \in \{U,V,W\}$ in the **MeshGridDecoderConfig** class (see subclause 5.2.2.2). The total number of resolution levels is equal to: $totalNumLevels = \max(nLevels.u, \max(nLevels.v, nLevels.w))$, where max returns the maximum absolute value between two numbers.

Given a "global" resolution level *globalLevel* of the mesh, which is defined in the range [0, *totalNumLevels* – 1], the direction-dependent resolution level is defined as:

$$level_d = \begin{cases} levelDiff_d, & levelDiff_d > 0 \\ 0, & levelDiff_d <= 0 \end{cases}, \quad where \quad levelDiff_d = globalLevel_d - (totalNumLevels - nLevels_d). \qquad (8)$$

The *displaylevel* field of the MeshGrid node (see subclauses 4.3.4.2.1 and 4.3.4.2.2) has to be specified as a "global" resolution level.

The number of reference-surfaces ($nSlices_d^l$) at direction-dependent resolution level l_d can be derived from the number of reference-surfaces ($nSlices_d^L$) at direction-dependent resolution level L_d , as illustrated in subclause 5.2.2.38.

B.1.2 Relation between the resolution level of the grid and the number of reference-surfaces.

For the reference-grid, the total number of resolution levels (*totalNumLevelsGrid*) is computed based on the maximum number of decomposition levels as follows.

The number of decomposition levels for each direction $d \in \{U,V,W\}$ can be computed using the following SDL code that accepts as input the number of slices stored in the *nSlices* field of the **MeshGridDecoderConfig** class (see subclause 5.2.2.2):

```
class DecLevel(int nSlices) {
  unsigned int nDecLevels = 0;
```

Copyright International Organization for Standardization © ISO/IEC 2006 All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  while (nSlices ≥ MIN_DEC_SIZE) {
    ++nDecLevels;
    nSlices = (nSlices + 1)/2;
  }
}
```

The maximum decomposition level (*maxDecLevel*) can be computed using the following SDL code:

```
class MaxDecLevel(PointUVW nSlices) {
  // the actual fields
  unsigned int value;

  // decomposition level of the slices number in direction u, v, w
  DecLevel decLevelU(nSlices.u);
  value = decLevelU.value;

  DecLevel decLevelV(nSlices.v);
  if (value < decLevelV.value)
    value = decLevelV.value;

  DecLevel decLevelW(nSlices.w);
  if (value < decLevelW.value)
    value = decLevelW.value;
}
```

The total number of resolution levels of the grid is defined as: $totalNumLevelsGrid = \max(maxDecLevel - 1, 1)$.

B.1.3 Region Of Interest (ROI): Computation of the ROIs size and position

The number of regions of interest ($nROIs_d^L$) at the highest resolution level ($L = N_L - 1$, where N_L is the number of resolution levels of the model) for each direction $d \in \{U,V,W\}$ is specified by the ***nROIs*** field of the ***MeshGridDecoderConfig*** class (see subclause 5.2.2.2). Further, the ROIs at the highest resolution level shall be distributed as explained in subclause B.1.3.1. For all lower resolution levels ($0 \le L < N_L - 1$) the number of ROIs ($nROIs_d^L$) shall be computed as shown in subclause 5.2.2.37, and their distribution shall be computed as illustrated in subclause B.1.3.2.

Notice that, the explanations below are generic for any direction $d \in \{U,V,W\}$, and the formulas given in the following subclauses have to be computed for each direction independently.

B.1.3.1 Distribution of the ROIs at the highest resolution level.

Given the number of ROIs ($nROIs^L = 2^k$, with $k = [0, \mathbb{R}^+)$ — see subclause C.3) and the number of slices ($nSlices^L$) at the highest resolution level ($L = N_L - 1$), the default size of the ROIs ($defSzROI$) is computed as:

$$defSzROI = \left\lfloor \frac{nSlices^L}{nROIs^L} \right\rfloor . \tag{9}$$

In the particular case when $nSlices^L$ is a multiple value of $defSzROI^L$, the size of each ROI ($szROI^L$) is equal to $defSizeROI^L$ and the start position of ROI idx ($posROI_{idx}^L$, with $0 \le idx < nROIs^L$), is computed as:

$$\begin{aligned} szROI_{idx}^L &= defSzROI^L \\ posROI_{idx}^L &= idx \bullet szROI_{idx}^L = idx \bullet defSzROI = idx \bullet \frac{nSlices^L}{nROIs^L} . \end{aligned} \tag{10}$$

In the general case $nSlices^L$ is not a multiple value of $nROIs^L$ and therefore the size of the ROIs will vary:

$$szROI^L \in \{defSzROI, defSzROI + 1\} . \tag{11}$$

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In the general case, in order to obtain a quasi uniform distribution of the ROIs, the size of each ROI and their position can be computed as described below:

First the remainder ($remNumSlices^L$) is calculated, which is used in the estimation of the fractional increment ($incr$). Secondly, the fractional increment is employed to compute the size ($szROI^L_{idx}$) and position ($posROI^L_{idx}$) of each ROI idx ($0 \le idx < nrROIs^L$), as shown in equation (12).

$$\begin{aligned} remNumSlices^L &= nSlices^L - defSzROI \bullet nROIs^L \\ incr &= \frac{remNumSlices^L}{nROIs^L} \\ prevIncr &= round(idx \bullet incr) \\ currIncr &= round((idx+1) \bullet incr) \\ szROI^L_{idx} &= defSzROI + currIncr - prevIncr \\ posROI^L_{idx} &= idx \bullet defSzROI + prevIncr \end{aligned} \tag{12}$$

where the operator $round$ does the rounding of a number.

To avoid any rounding errors due to the limited precision of floating point numbers, equation (12) can be written using integer mathematics as follows:

$$\begin{aligned} remNumSlices^L &= nSlices^L - defSzROI \bullet nROIs^L \\ halfNumROIs &= \frac{nROIs^L}{2} \\ prevIncr &= \left\lfloor \frac{idx \bullet remNumSlices^L + halfNumROIs}{nROIs^L} \right\rfloor \\ currIncr &= \left\lfloor \frac{(idx+1) \bullet remNumSlices^L + halfNumROIs}{nROIs^L} \right\rfloor \\ szROI^L_{idx} &= defSzROI + currIncr - prevIncr \\ posROI^L_{idx} &= idx \bullet defSzROI + prevIncr \end{aligned} \tag{13}$$

where and $\lfloor v \rfloor$ returns the integer part of v .

The formulas from equation (13) shall be used to compute the size and position of the ROIs at the highest resolution level of the mesh. One notices that if $remNumSlices^L = 0$ than $szROI_i^{L-1}$ and $posROI_i^{L-1}$ yield the same values as computed with equation (10).

B.1.3.2 Distribution of the ROIs at the lower resolution levels.

In order to allow a seamless display of the contents belonging to ROIs of different resolution levels, there must be established an appropriate correspondence between the ROIs positions and the sizes of different resolution levels. Hence the ROIs of different resolution levels shall be hierarchically nested.

Between the number of ROIs of a resolution level L ($nROIs^L$) and the number of ROIs of the immediate higher resolution level $L+1$ ($nROIs^{L+1}$) are possible the following ratios:

$$nROIs^L = \begin{cases} \text{(a) } nROIs^{L+1}, & \text{if } nROIs^{L+1} = 1 \\ \text{(b) } nROIs^{L+1}/2, & \text{if } nROIs^{L+1} = 2^k \end{cases} \tag{14}$$

According to equation (14), $nROIs^L$ shall be kept the same as $nROIs^{L+1}$ (case (a)) only when $nROIs^{L+1} = 1$. Otherwise (case (b)) $nROIs^L = nROIs^{L+1}/2$, i.e. the number of ROIs shall be divided by two with each lower resolution level, such that $szROI^L \in \{defSzROI, defSzROI + 1\}$.

© ISO/IEC 2006 – All rights reserved

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

The size and position of each ROI at any resolution level (L) based on the distribution of the ROIs at the highest resolution level ($N_L - 1$) can be computed with an iterative approach, as described below:

As shown in equation (15), the ratio ($ratio^L$) between the number of ROIs of consecutive resolution levels ($L+1$ and L) can be used to derive the size ($szROI^L_{idx}$) and position ($posROI^L_{idx}$) of each ROI (idx with $0 \leq idx < nrROIs^L$) of resolution level (L) with respect to corresponding ROIs (j , k) of resolution level ($L+1$).

$$
\begin{aligned}
& j = (idx \bullet ratio^L) \text{ and } k = (j + ratio^L - 1) \text{ with } ratio^L = \frac{nROIs^{L+1}}{nROIs^L} \\
& posROI^L_{idx} = \left\lfloor \frac{posROI^{L+1}_j + 1}{2} \right\rfloor \text{ and } endPosROI^L_{idx} = \left\lfloor \frac{posROI^{L+1}_k + szROI^{L+1}_k - 1}{2} \right\rfloor \\
& szROI^L_{idx} = endPosROI^L_{idx} - posROI^L_{idx} + 1
\end{aligned}
\tag{15}
$$

In equation (15) $endPosROI^L_{idx}$ is the end position of ROI (idx) and $\lfloor v \rfloor$ returns the integer part of v . Finally, to estimate the size and position of each ROI at any resolution level (L) with respect to the corresponding ROIs at the highest resolution level ($N_L - 1$) directly, one can generalize equation (15) and obtain the result from equation (16).

$$
\begin{aligned}
& j = (idx \bullet ratio^L) \text{ and } k = (j + ratio^L - 1) \text{ with } ratio^L = \frac{nROIs^{N_L - 1}}{nROIs^L} \text{ and } \Delta l = N_L - L - 1 \\
& posROI^L_{idx} = \left\lfloor \frac{posROI^{N_L - 1}_j + 2^{\Delta l} - 1}{2^{\Delta l}} \right\rfloor \text{ and } endPosROI^L_{idx} = \left\lfloor \frac{posROI^{N_L - 1}_k + szROI^{N_L - 1}_k - 1}{2^{\Delta l}} \right\rfloor \\
& szROI^L_{idx} = endPosROI^L_{idx} - posROI^L_{idx} + 1
\end{aligned}
\tag{16}
$$

Notice in equation (16) that the indices (j , k) are computed for the highest resolution level.

B.2 Scalability Modes

The MeshGrid multi-resolution representation allows three types of scalability modes such as: (1) *mesh resolution scalability*, i.e. adapting the number of transmitted vertices, (2) *shape precision* i.e. adaptive reconstruction of the reference-grid positions, and (3) *vertex position scalability*, i.e. increasing the precision of known vertex positions with relation to the reference-grid (the value of the offset). Each of these scalability modes is possible at the level of the ROI used for coding the MeshGrid (see subclause 5.1).

B.2.1 The mesh resolution scalability

The *mesh resolution scalability* is illustrated in Figure B.3 for three different resolutions of an object. Different mesh resolutions represent different connectivity-wireframe descriptions of the same object (see B.1). The first row of images shows a shaded surface of its polygonal representation with the outline of the connectivity-wireframe drawn on top of the shaded surface, while the second row of images display the object as a wireframe of the polygonized representation.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

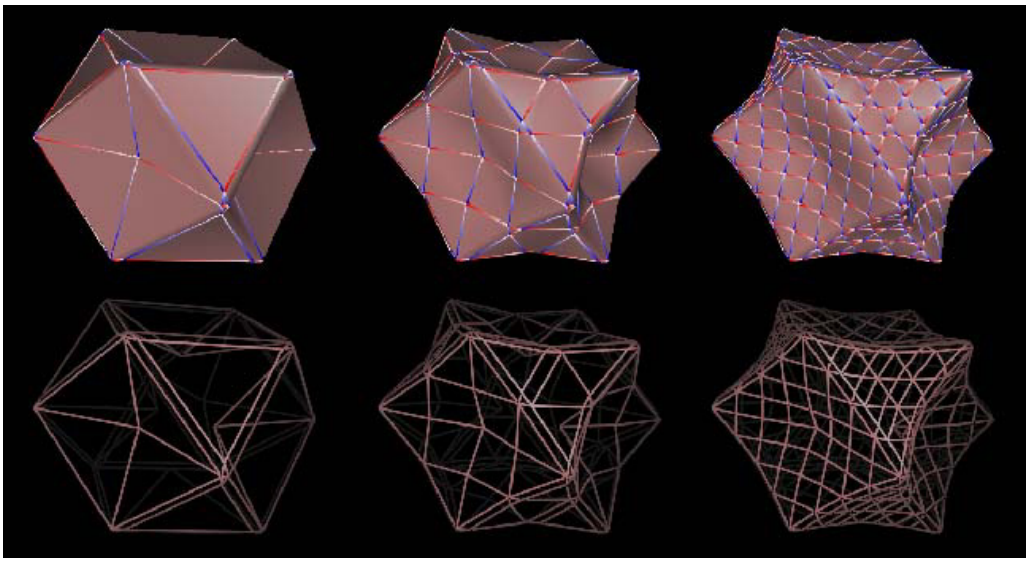
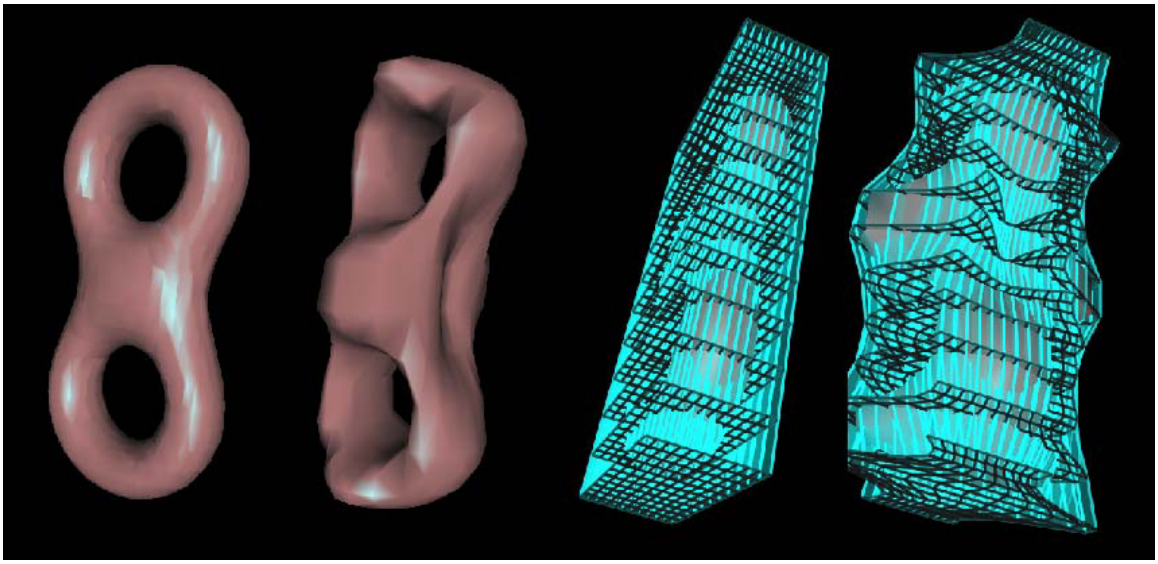


Figure B.3 — Scalability: the mesh resolution is increasing from left to right.

B.2.2 The scalability in shape precision

The *scalability in shape precision* is supported by the reference-grid. The reference-grid contains the description about the volume of the object, the vertices of the connectivity-wireframe being attached to it, as explained in subclause 4.3.4.3.2. This implies that the 3D distribution of the vertices for a certain mesh resolution level (the connectivities are kept unchanged), is given by the distribution of the reference-grid points to which vertices are attached to.

The minimal description for a reference-grid consists of the coordinates of the eight corners of the reference-grid that in general define a non-regular deformed box. For this minimal description the obtained reference-grid is uniformly distributed (see Figure B.4 (a), (c)).



(a) (b) (c) (d)

Figure B.4 — Progressively refining the reference-grid

© ISO/IEC 2006 – All rights reserved

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

This scalability aspect can be exploited for progressively refining non-uniform distributed reference-grid points (see Figure B.4 (b), (d)). Another example is shown in Figure B.5. The coordinates of the reference-grid points are progressively refined from left (Figure B.5(a)), to right (Figure B.5(d)). The first row of images show a shaded surface of its polygonal representation with the outline of the connectivity-wireframe drawn on top of the shaded surface, while the second row of images display the object as a wireframe of the polygonized representation.

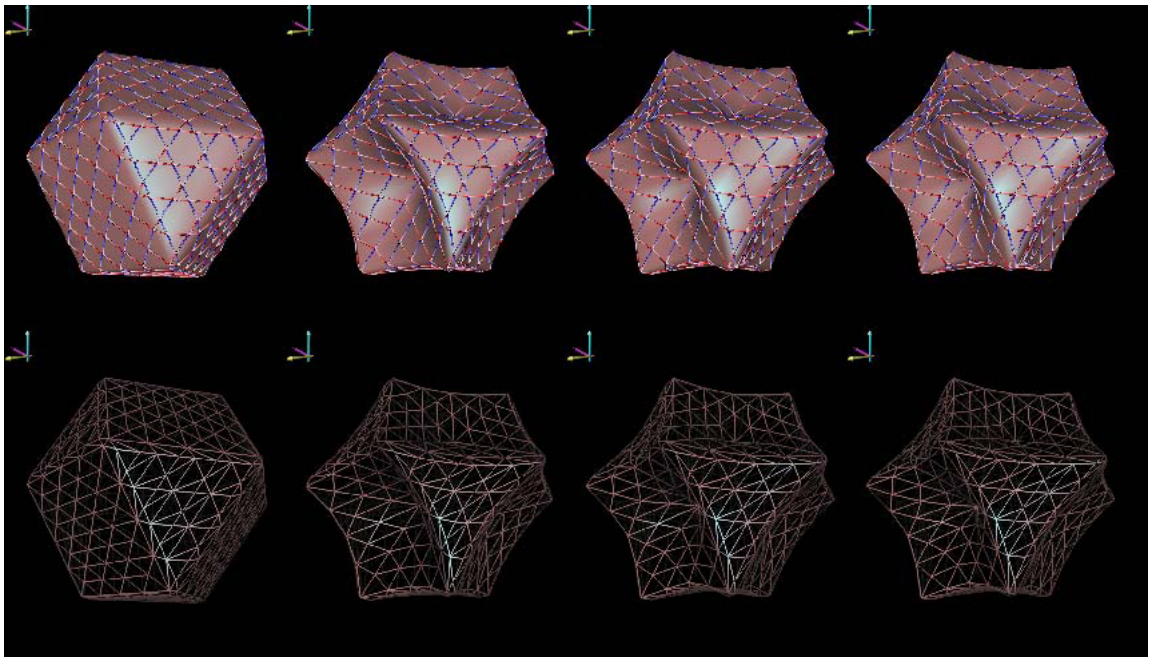


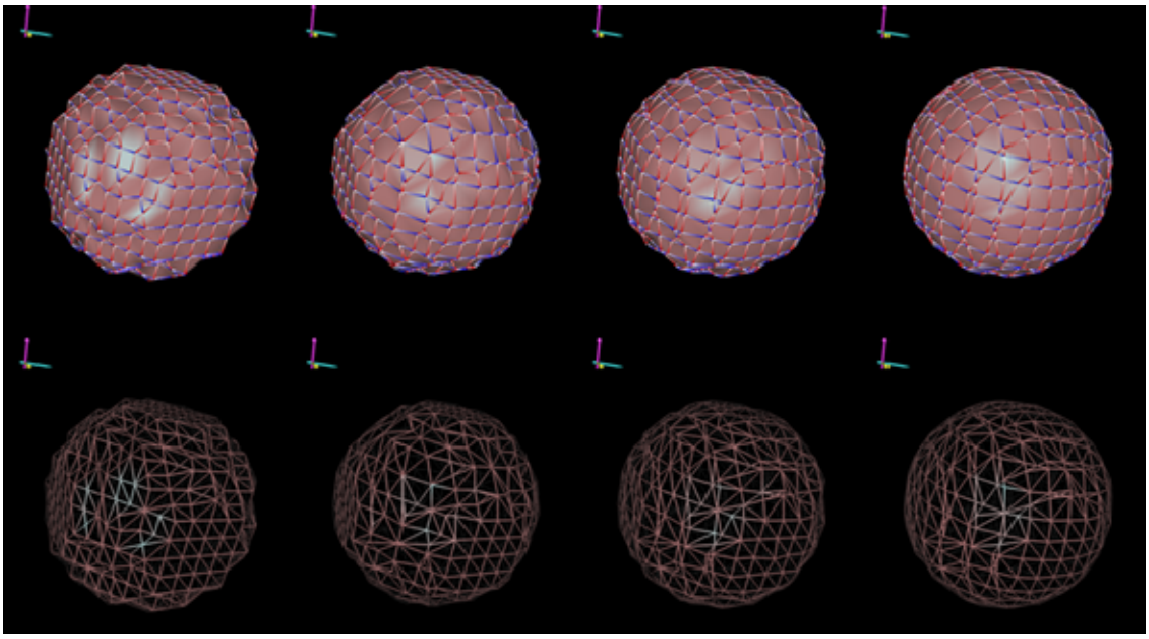
Figure B.5 — Scalability in shape precision

B.2.3 The vertex position scalability

The vertices of the connectivity-wireframe have one degree of freedom relative to the reference-grid points, i.e. the offset value is a relative value (see subclause 4.3.4.3.2) that can vary between [0,1).

When generating the stream for a MeshGrid surface description, the vertices' offset values are quantized on a specified number of bits (see subclause 5.2.2.23), and are encoded bit-plane by bit-plane as discussed in subclause 5.2.3.3. When receiving the stream, the vertices offset values are refined, using the progressive quantization approximation scheme, bit-plane by bit-plane. The default offset value is 0.5, which is set when the offset has no value specified yet.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



0 bits 1 bits 2 bits 5 bits

Figure B.6 — Vertex position scalability

The progressive refinement of the vertices offset values is illustrated in Figure B.6 for a sphere. The first row of images show a shaded surface of its polygonal representation of the sphere with the outline of the connectivity-wireframe drawn on top of the shaded surface, while the second row of images display the sphere as a wireframe of the polygonized representation. The result for a default offset (0.5) is shown in Figure B.6(a); notice the local effect due to discretisation. The refinement of the vertices offset initially quantized on 5 bits, is illustrated for an increasing number of refinement bits from left, Figure B.6(b), to right, Figure B.6(d).

B.3 Animation Possibilities

The MeshGrid representation allows in addition to the vertex-based animation typical for IndexedFaceSet represented objects, specific animation possibilities, such as (1) rippling effects by changing the position of the vertices relative to corresponding reference-grid points and (2) reshaping the regular reference-grid. The latter form of animation can be done on a hierarchical multi-resolution basis (see subclause 4.3.4.3.1: i.e. deforming the reference-grid for a lower resolution mesh, will propagate the action to the higher levels, while applying a similar deformation at the higher resolution levels will only have a local impact. The vertices of the wireframe will be updated, each time the grid point, it is attached to, moves (see subclause 4.3.4.3.2).

B.3.1 Vertex-based animation

Vertex animation applies to the vertices of the connectivity-wireframe. This type of animation requires that each vertex retrieves its initial coordinates from the grid and its associated offset, and stores, updates these coordinates during the animation, without further referencing the grid.

B.3.2 Rippling effects, changing the offsets

Changing the vertices' offsets allows limited movements, oscillations (rippling effects) around the reference-grid points they are attached to (see subclause 4.3.4.3.2). Such an example is given in Figure B.7 (the sphere). Both images show different frames during the animation. This type of animation is easier to perform if all the vertices' offsets have the same value, preferably equal to 0.5. Obtaining a constant offset of 0.5 for all the vertices, as illustrated in Figure C.1, is feasible by positioning the boundary grid positions in such a way that the vertices lie at half the distance in between two boundary grid positions.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved

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

If those offsets are not equal to 0.5 everywhere, this might give artifacts, e.g. for an overall vertex movement of 0.3, vertices already having an offset of 0.8 will get an offset equal to 1.1, exceeding the theoretical accepted range of [0, 1), which might give self intersections of the mesh triangles.

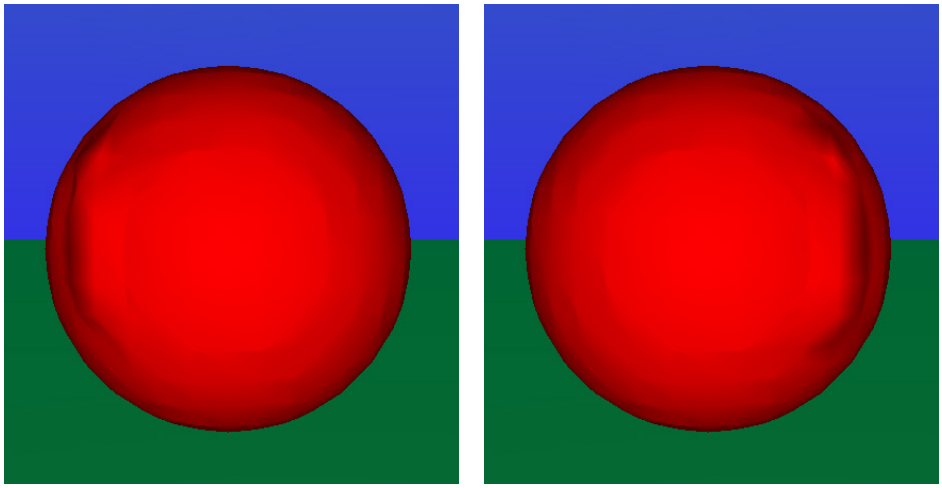
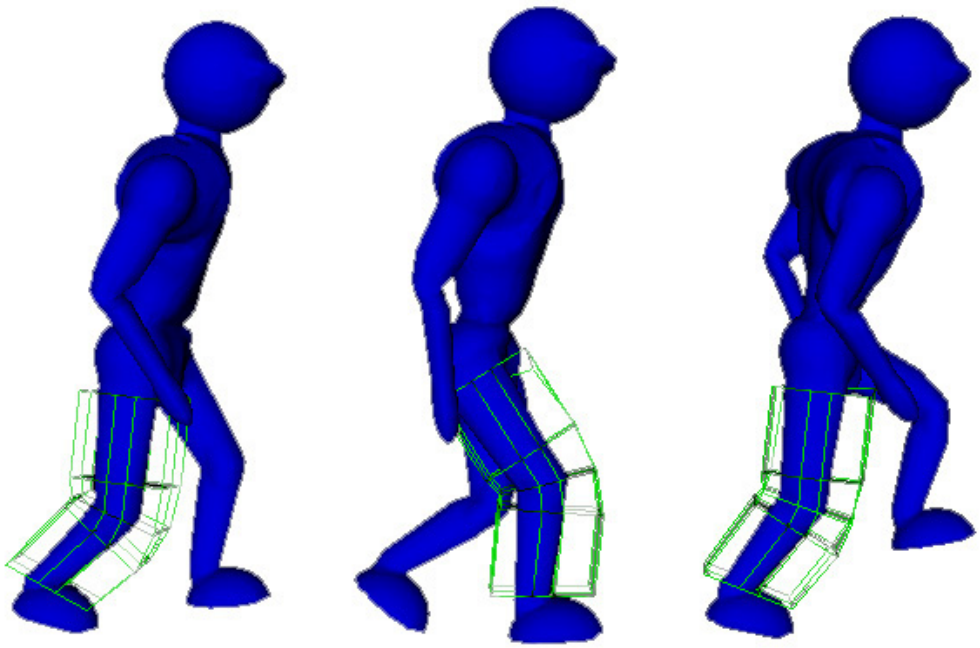


Figure B.7 — Shows two frames from a sequence of animations of a sphere: rippling effects.

B.3.3 Animation of the reference-grid points

Animating or deforming the reference-grid points will affect the global shape of the object. This type of animation will not change the connectivity between the vertices; it will only change the coordinates of the vertices when they are updated from the corresponding reference-grid positions (see subclause 4.3.4.3.1).

An example of a sequence of frames obtained by animating the reference-grid of a humanoid model is illustrated in Figure B.8(a)-(c). A part of the reference-grid is displayed together with the shaded surface, in order to illustrate the impact of the animation on the mesh and grid. Another example of reference-grid animation is the morphing of a face as illustrated in the sequence of images from Figure B.9.



(a) (b) (c)

Figure B.8 — Reference-grid animation, using Skin&Bones system, of the humanoid model

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2006 – All rights reserved

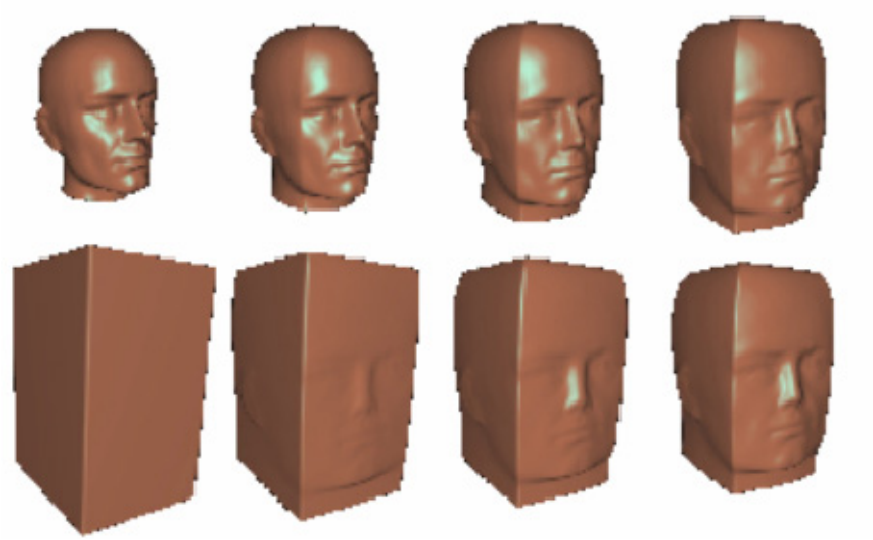


Figure B.9 — Morphing of a human head by deforming the reference-grid

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C
(informative)

MeshGrid representation

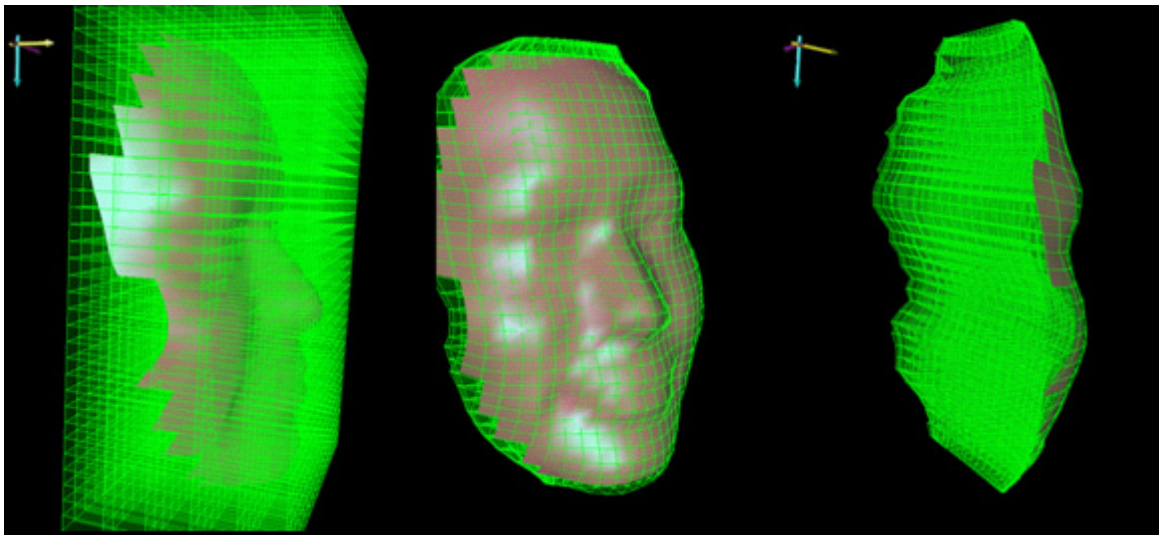
C.1 Representing Quadrilateral meshes in the MeshGrid format

A quadrilateral mesh has the property that each vertex has 4 neighbors. The MeshGrid representation has the same constraint on the vertices' connectivities as the quadrilateral meshes. A quadrilateral mesh is in fact a subset of the possible mesh representations supported by the MeshGrid format, since the latter may combine: triangles, quadrilaterals, pentagons, hexagons and heptagons.

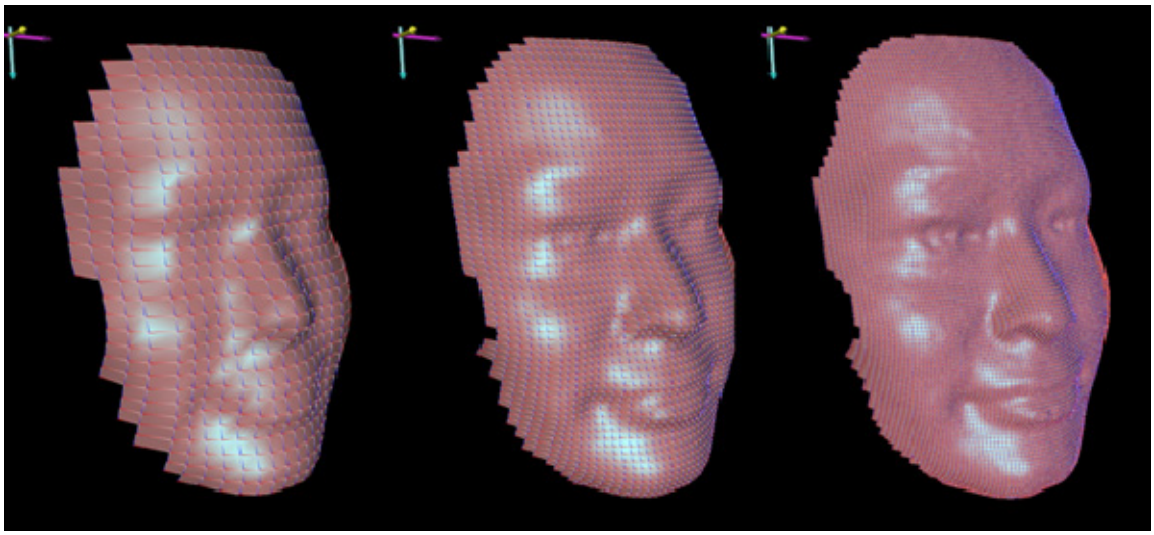
There are three steps involved in the representation of quadrilateral meshes of type1 (see classification from subclause 5.2.3.5) in the MeshGrid format: (1) defining the constrained reference-grid (see Figure C.1 (a)), (2) deforming and distributing the reference-grid to fit the quadrilateral mesh (see Figure C.1 (b)-(c)), and (3) deriving the connectivity-wireframe from the quadrilateral mesh and attaching it to the reference-grid (see Figure C.1 (d)-(f) displaying the multi-resolution MeshGrid).

The original single-resolution quadrilateral mesh illustrated in Figure C.1 has been brought a multi-resolution MeshGrid model.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



(a) (b) (c)

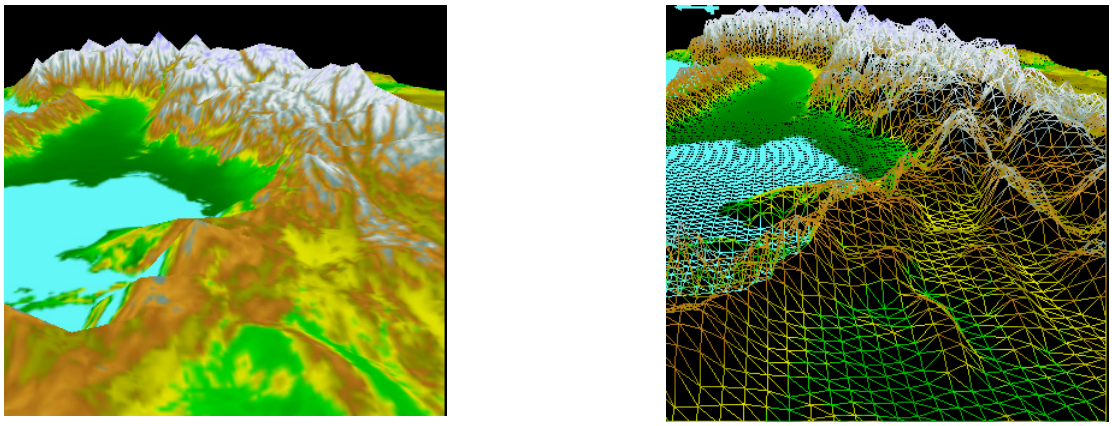


(d) (e) (f)

Figure C.1 — A spherical mapped quadrilateral mesh obtained from a structured light scanner, visualized at different resolutions of the MeshGrid representation

© ISO/IEC 2006 – All rights reserved

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

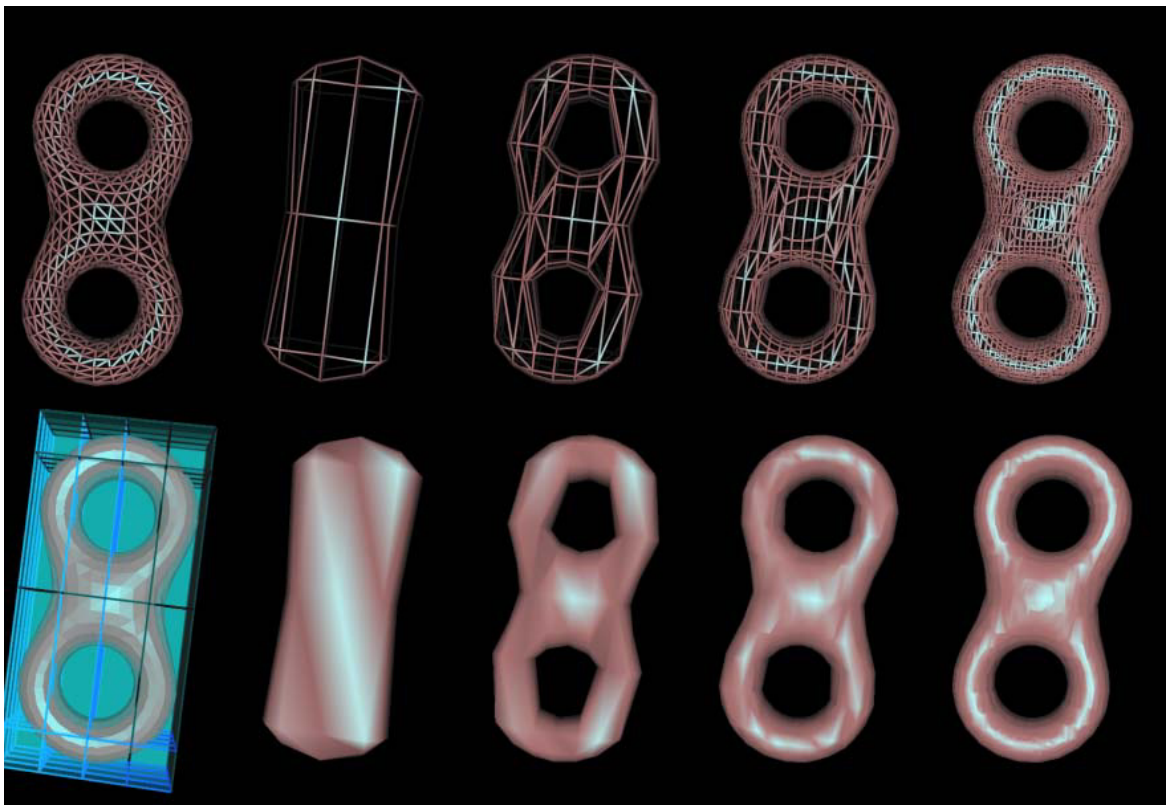


(a) (b)

Figure C.2 — Example of a quadrilateral mesh with texture mapping.

C.2 IndexedFaceSet models represented by the MeshGrid format.

Arbitrary IndexedFaceSet models have to be remeshed in order to be brought to the MeshGrid format. Figure C.3 shows the multi-resolution MeshGrid model obtained from a single-resolution IndexedFaceSet surface. Notice that topological changes are allowed between successive resolutions of the MeshGrid model.



(a) (b) (c) (d) (e)

Figure C.3 — Remeshing of an eight model. Initial representation is an IndexedFaceSet (a), converted to a multi-resolution MESHGRID (b till e). The Reference-Grid has been set non-equidistant to better encode the topology and shape of the model.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

C.3 Computation of the number of ROIs at the highest resolution level given an optimal ROI size

Given an optimal size of the ROI ($szROI_{Optimal}$), which is estimated according to the memory constraints of the target device, the optimal number of ROIs ($nROIs_{Optimal}$) is computed as: $nROIs_{Optimal} = \frac{nSlices}{szROI_{Optimal}}$, where $nSlices$ is the number of slices. When dealing with a multi-resolution model of N_L resolution levels, $nROIs_{Optimal}$ shall be computed for the highest resolution level. Furthermore, to preserve the same optimal size of the ROI for each resolution level L ($0 \leq L \leq (N_L - 1)$), since $nSlices^L \simeq \frac{nSlices^{L+1}}{2}$, an appropriate value for $nROIs_{Optimal}$ is 2^k , with $k = [0, \mathbb{R}^+)$. In general the value of $nROIs_{Optimal} \neq 2^k$, and $nROIs_{Optimal}$ will lie in the interval $[2^k, 2^{k+1}]$. Considering the above constraint, the actual number of ROIs ($nROIs$) would have to be set to either 2^k or 2^{k+1} , depending on which value produces an ROI size ($szROI$) that is closer to the optimal ($szROI_{Optimal}$).

Notice that the number of ROIs is encoded in the **nROIs** field of the **MeshGridDecoderConfig** class (see subclause 5.2.2.2).

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

© ISO/IEC 2006 – All rights reserved

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

Annex D
(informative)

Solid representation

D.1 Overview

Solid representation functionalities deal with:

— reducing the size of files to be transmitted or shared by transmitting constructive commands rather than results;

— increasing the rendered objects' geometrical precision by managing a polynomial representation of volumes based on implicit primitives;

— manipulating complex objects combined by solid operations. The algebraic representation is extended to the Arithmetic of Forms.

D.2 Solid Primitives

Any solid object has a three-dimensional form defined by a skin, which delimits the inside from the outside. Each point of this form can be deducted according to a mathematical formula. If the shape equation is unknown or impossible to manipulate, it will be sectioned in smaller pieces until they arrive at known forms that we will call primitives. These primitives will then be assembled in order to constitute the original complex shape.

There are two approaches to define a surface. The first one is to usea parametric equation, which makes it possible to calculate the spatial coordinates of each surface point.

The second one is to use algebraic surfaces represented by an implicit equation, which makes it possible to know if an unspecified point in space P is inside, on or outside the volume delimited by this surface.

For example, the implicit equation of a sphere with radius R and centered at C is as follows:

(Px-Cx) **2** + (Py-Cy) **2** + (Pz-Cz) **2** – R **2** = 0

Thus for each point in space, if the result of the equation is < 0 it is inside, = 0, it will be on and > 0, it will be outside.

In this approach, using the general form of an implicit equation we can represent a very broad, yet unexplored, range of algebraic surfaces.

A quadratic equation (second degree equation), allows to define the entire quadrics family is covered, while an equation of the fourth degree allows to define the quartics family.

Because of these possibilities, implicit forms are well suited to the transmission of solid object representation in exact geometry.

The implicit equation of the second degree that defines quadrics, is as follows:

$$c_0 X_0^2 + c_1 X_0 X_1 + c_2 X_1^2 + c_3 X_0 X_2 + c_4 X_1 X_2 + c_5 X_2^2 + c_6 X_0 X_3 + c_7 X_1 X_3 + c_8 X_2 X_3 + c_9 X_3^2 = 0$$

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

where, for each point coordinates (X0,X1,X2,X3), the result is

$$F(X_0,..., X_3) < 0$$
$$F(X_0,..., X_3) = 0$$
$$F(X_0,..., X_3) > 0$$

whether the point is inside, on the surface or outside.

In the above equation, the point coordinates were made homogeneous by the addition of a fourth coordinate X3.

A surface of the second degree thus requires 10 coefficients; a surface of the fourth degree (quartic) requires 35 of them. The initialization of these coefficients makes it possible to numerically define any algebraic surface.

This surface can extend to infinity. For example the cylinder equation represents a cylinder that prolongs to infinity at both its extremities. It is therefore important to bind it (to circumscribe it by finite space) so that is can be processed and displayed.

In 3D modeling, using algebraic surfaces by coefficients is difficult because there is no direct obvious link between these coefficients and the resulting geometry.

In the field of projective geometry, an incidence-based geometry without measurement or parallelism, there are construction mechanisms that allow defining curves, surfaces and volumes from geometrical properties. These construction mechanisms were used as a starting point to resolve algebraic equations, allowing coefficient deduction of the corresponding quadric's algebraic equation.

D.3 Arithmetics of Forms

The solid representation is based on a logical modeling system for solid objects called "Design Logic" [66], [67]. The user combines the geometric primitives with the help of solid operators that enables to specify both the nature of the operations, such as adding or removing matter, and their scheduling.

The description of a solid object takes the form of a solid tree made up of operators that will be described below, and operands that may be either elementary primitives such as a sphere or cylinder, or more complex solid structures. In the first case we talk about the leaves of a volume tree and in the second case about solid sub-trees that display all the characteristics of a general tree.

The first concept to remember in Design Logic is density. In fact, every basic geometric primitive splits space into three regions - external, boundary and internal - coded by the integers 0, 1, 2. This initial ternary coding of space tells us the density of every point in space with respect to any elementary primitive. Using the implicit functions' property of these primitives that verifies if a point of space is inside, outside or on the surface, it is possible to establish the ternary logic according to the function result.

For example, every point inside a quadric will have density 2 with respect to that quadric. Given a space thus arithmetized, we are able to combine primitives or elementary volumes with the help of arithmetic operators. This creates the capabilities to manage matter that is made up of volumes.

A complex volume consists of cell solid units assembled with the help of solid operations and producing a solid tree.

The most common operations are those used by the CSG (Constructive Solid Geometry), union, intersection and logical subtraction. These operations were extended to arithmetic operators applied on densities of an object [75].

These operations can easily be implemented for implicit-defined volumes as well as for volumes whose surface is polygonized, as long as their implicit attributes have been preserved (by an inclusion in a **SolidRep** node, for example).

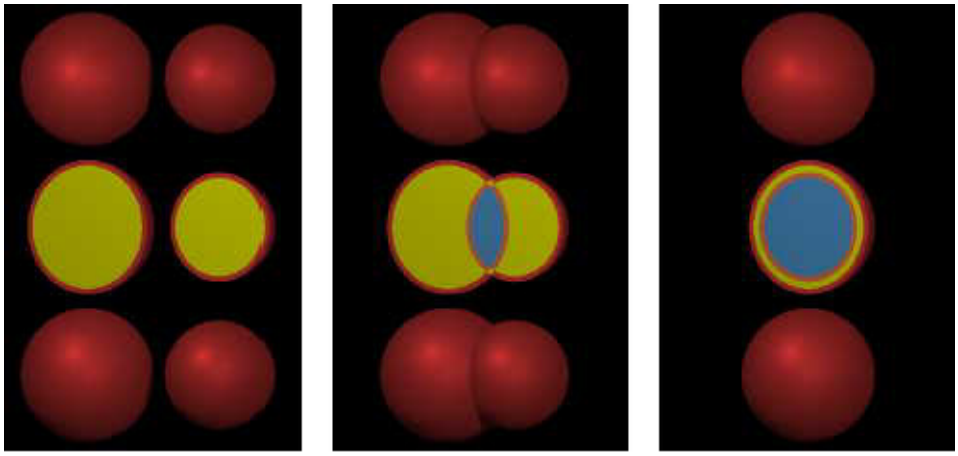
© ISO/IEC 2006 – All rights reserved

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

An operation between two volumes is applied for each point in space and is independent of the final representation mode: for each point in space, the result is the density obtained from the operation carried out on these volumes' densities.

D.3.1 General Arithmetic Operators on Densities

The basic set of operators, of purely arithmetic origin, achieve combinations of solid forms by isolating or selecting certain regions in a combinatorial way by means of their densities. These operators are for instance addition, multiplication and difference of the densities of two volumes.



Addition

The illustration is a series of three figures:

— The left figure represents two distinct spheres.

— The center figure represents the two spheres that intersect.

— The right figure represents the two overlapping spheres.

Each figure is made up of three pictures:

— The top picture indicates the two spheres' respective positions.

— The center picture is the cross-section result of the addition operation.

— The bottom picture shows the integral result of the operation.

A density is associated to each color:

0 black

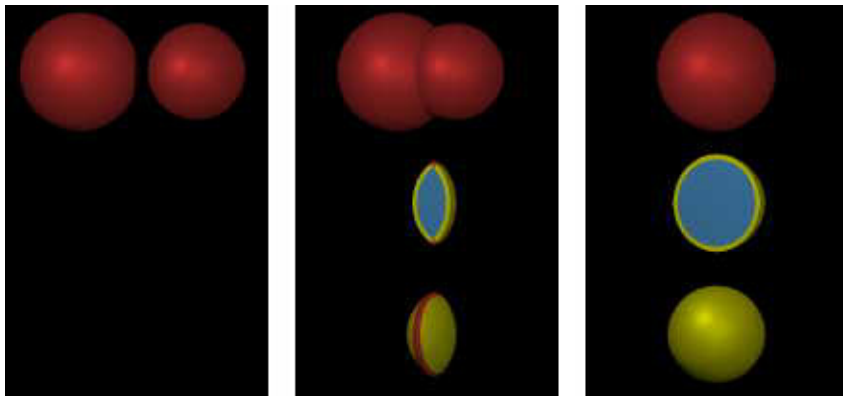
1 red

2 yellow

3 orange

4 blue

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



Multiplication

The illustration is a series of three figures:

— The left figure represents two separate spheres.

— The center figure represents the two spheres that intersect.

— The right figure represents the overlapping spheres.

Each figure is made up of three pictures:

— The top picture indicates the spheres' respective positions.

— The center picture represents the cross-section result of the product operation.

— The bottom picture shows the integral result of the operation.

A density is associated to each color:

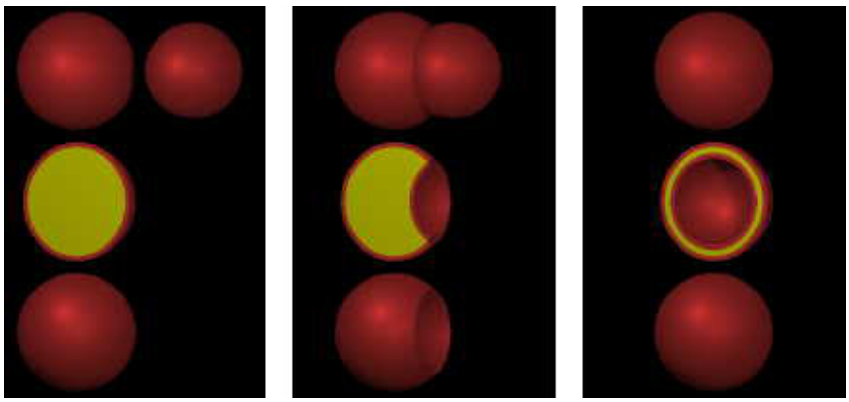
0 black

1 red

2 yellow

3 orange

4 blue



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

© ISO/IEC 2006 – All rights reserved

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

Difference

The illustration is a series of three figures:

— The left figure represents two distinct spheres.

— The center figure represents the two spheres that intersect.

— The right figure represents the two overlapping spheres.

Each figure is made up of three pictures:

— The top picture indicates the two spheres' respective positions.

— The center picture is the cross-section result of the difference operation.

— The bottom picture shows the integral result of the operation.

A density is associated to each color:

0 black

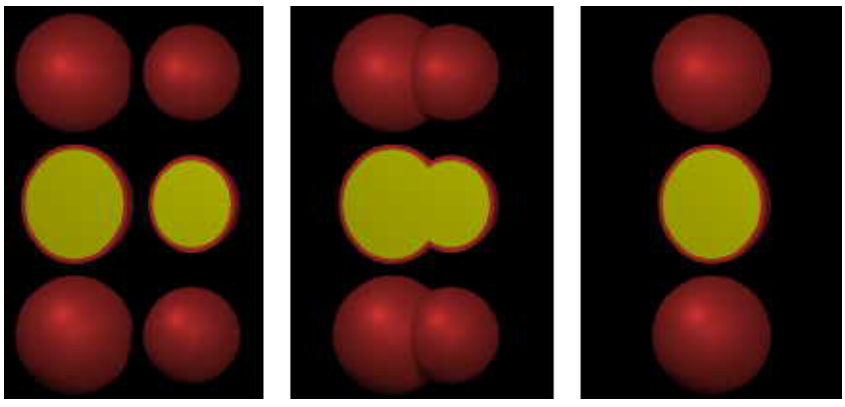
1 red

2 yellow

D.3.2 Ternary Logic : The Kleene Operators

The second set of operators use the ternary logic only and, in many cases, are sufficient to carry out complex solid modeling. Four of these operators have been directly borrowed from the system of ternary logic established by S. Kleene in 1952 [45]. The user familiar with the CSG systems will find that Boolean operators are binary subcases of the ternary operators. But the latter avoid special cases introduced by the way CSG handles boundaries, and they may at any time benefit from the possibility of getting access to the densities of the regions for more sophisticated solid operations.

These operators are for instance the ternary union and the ternaryintersection of two volumes.



ternary union

The illustration is a series of three figures:

— The left figure represents two distinct spheres.

— The center figure represents the two spheres that intersect.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— The right figure represents the two overlapping spheres.

Each figure is made up of three pictures:

— The top picture indicates the two spheres' respective positions.

— The center picture represents the cross-section result of the ternary union operation.

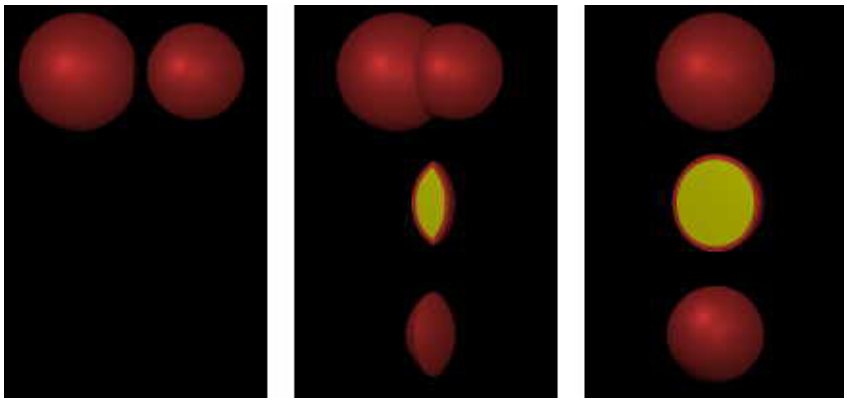
— The bottom picture shows the integral result of the operation.

A density is associated to each color:

0 black

1 red

2 yellow



ternary intersection

The illustration is a series of three figures:

— The left figure represents two separate spheres.

— The center figure represents the two spheres that intersect.

— The right figure represents the overlapping spheres.

Each figure is made up of three pictures:

— The top picture indicates the spheres' respective positions.

— The center picture represents the cross-section result of the ternary intersection operation.

— The bottom picture shows the integral result of the operation.

A density is associated to each color:

0 black

1 red

2 yellow

© ISO/IEC 2006 – All rights reserved

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

D.3.3 Densities Filtering

Systematic exclusion of matter with density zero may enable to avoid any filtering. For example, visualization of the surface and interior of a quadric follows from the simple preservation of densities 1 and 2, and discarding density 0 by convention.

Nevertheless, when the combinatorial logic of the designer requires precise control on transition states of the numerical value of the density, we may need a totally explicit filtering.

In this case the multivalued nature of the different regions in space allows for a large variety of arithmetic selections of a single volume object.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex E
(informative)

Face and Body animation: XMT compliant animation and encoding parameter file format

E.1 XSD for FAP file format

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="urn:mpeg:mpeg4:fap:schema:2002" xmlns:fap="urn:mpeg:mpeg4:fap:schema:2002"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <annotation>
        <documentation>
            Face Animation Parameters Textual format Schema
        </documentation>
    </annotation>
    <annotation>
        <documentation> basic types </documentation>
    </annotation>
    <simpleType name="ZeroOneType">
        <restriction base="integer">
            <minInclusive value="0"/>
            <maxInclusive value="1"/>
        </restriction>
    </simpleType>
    <simpleType name="ListZeroOneType">
        <list itemType="fap:ZeroOneType"/>
    </simpleType>
    <simpleType name="FaceAnimationParameterType">
        <restriction base="integer"/>
    </simpleType>
    <annotation>
        <documentation>
        ***************************************************************
        *                                                             *
        * Declaration of the top-level containers (FAP, Header, ...)  *
        *                                                             *
        ***************************************************************
        </documentation>
    </annotation>
    <element name="FAP">
        <complexType>
            <sequence>
                <element ref="fap:FaceAnimationHeader" minOccurs="0"/>
                <element name="body" minOccurs="0">
                    <complexType>
                        <sequence>
                            <element ref="fap:FaceFrameData" minOccurs="0" maxOccurs="unbounded"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
    <element name="FaceAnimationHeader">
        <complexType>
            <attribute name="version" type="float" use="optional" default="3.0"/>
            <attribute name="name" type="string" use="required"/>
            <attribute name="frameRate" type="unsignedInt" use="optional" default="10"/>
            <attribute name="frameCount" type="unsignedInt" use="required"/>
        </complexType>
    </element>
    <annotation>
```

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

© ISO/IEC 2006 – All rights reserved

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

```
        <documentation>
        ***********************************************************
        *                                                         *
        * Declaration of the dynamic containers (FaceFrameData)   *
        *                                                         *
        ***********************************************************
        </documentation>
    </annotation>
    <element name="FaceFrameData">
        <complexType>
            <attribute name="faceFrameMask" use="required">
                <simpleType>
                    <restriction base="fap:ListZeroOneType">
                        <length value="68"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="frameIndex" type="unsignedInt" use="required"/>
            <attribute name="faceFrameValues">
                <simpleType>
                    <list>
                        <simpleType>
                            <restriction base="fap:FaceAnimationParameterType">
                                <minLength value="0"/>
                                <maxLength value="68"/>
                            </restriction>
                        </simpleType>
                    </list>
                </simpleType>
            </attribute>
        </complexType>
    </element>
</schema>
```

E.2 XSD for BAP file format

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="urn:mpeg:mpeg4:bap:schema:2002" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:bap="urn:mpeg:mpeg4:bap:schema:2002" elementFormDefault="qualified">
    <annotation>
        <documentation>
            Body Animation Parameters Textual format Schema
        </documentation>
    </annotation>
    <annotation>
        <documentation> basic types </documentation>
    </annotation>
    <simpleType name="ZeroOneType">
        <restriction base="integer">
            <minInclusive value="0"/>
            <maxInclusive value="1"/>
        </restriction>
    </simpleType>
    <simpleType name="ListZeroOneType">
        <list itemType="bap:ZeroOneType"/>
    </simpleType>
    <simpleType name="BodyAnimationParameterType">
        <restriction base="integer">
            <minInclusive value="-314159"/>
            <maxInclusive value="314159"/>
        </restriction>
    </simpleType>
    <annotation>
        <documentation>
        ***************************************************************
        *                                                             *
        * Declaration of the top-level containers (BAP, header, ...)  *
        *                                                             *
        ***************************************************************
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2006 – All rights reserved

```
        </documentation>
    </annotation>
    <element name="BAP">
        <complexType>
            <sequence>
                <element ref="bap:BodyAnimationHeader" minOccurs="0"/>
                <element name="body" minOccurs="0">
                    <complexType>
                        <sequence>
                            <element ref="bap:BodyFrameData" minOccurs="0" maxOccurs="unbounded"/>
                        </sequence>
                    </complexType>
                </element>
            </sequence>
        </complexType>
    </element>
    <element name="BodyAnimationHeader">
        <complexType>
            <attribute name="version" type="float" use="optional" default="3.0"/>
            <attribute name="name" type="string" use="required"/>
            <attribute name="frameRate" type="unsignedInt" use="optional" default="10"/>
            <attribute name="frameCount" type="unsignedInt" use="required"/>
        </complexType>
    </element>
    <annotation>
        <documentation>
        ********************************************************
        *                                                      *
        * Declaration of the dynamic containers (BodyFrameData)   *
        *                                                      *
        ********************************************************
        </documentation>
    </annotation>
    <element name="BodyFrameData">
        <complexType>
            <attribute name="bodyFrameMask" use="required">
                <simpleType>
                    <restriction base="bap:ListZeroOneType">
                        <length value="296"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="frameIndex" type="unsignedInt" use="required"/>
            <attribute name="bodyFrameValues">
                <simpleType>
                    <list>
                        <simpleType>
                            <restriction base="bap:BodyAnimationParameterType">
                                <minLength value="0"/>
                                <maxLength value="296"/>
                            </restriction>
                        </simpleType>
                    </list>
                </simpleType>
            </attribute>
        </complexType>
    </element>
</schema>
```

E.3 XSD for EPF

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:epf="urn:mpeg:mpeg4:epf:schema:2002"
targetNamespace="urn:mpeg:mpeg4:epf:schema:2002" elementFormDefault="qualified">
    <annotation>
        <documentation>
            Face and Body Parameters Encoding Textual format Schema
        </documentation>
    </annotation>
    <annotation>
```

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

© ISO/IEC 2006 – All rights reserved

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

```
    <documentation> basic types </documentation>
</annotation>

<simpleType name="ZeroOneType">
    <restriction base="integer">
        <minInclusive value="0"/>
        <maxInclusive value="1"/>
    </restriction>
</simpleType>

<simpleType name="ListZeroOneType">
    <list itemType="epf:ZeroOneType"/>
</simpleType>

<simpleType name="FrameRateType">
    <restriction base="integer"/>
</simpleType>


<element name="EPFFrame">
    <complexType>
        <attribute name="FBM" use="required">
        <simpleType>
            <restriction base="integer">
                <minInclusive value="0"/>
                <maxInclusive value="3"/>
            </restriction>
        </simpleType>

        </attribute>
        <attribute name="FFR" use="optional">
            <simpleType><restriction base="integer">
                <minInclusive value="0"/>
                <maxInclusive value="99999"/>
            </restriction></simpleType>
        </attribute>
        <attribute name="BFR" use="optional">
            <simpleType><restriction base="integer">
                <minInclusive value="0"/>
                <maxInclusive value="99999"/>
            </restriction></simpleType>
        </attribute>
        <attribute name="ISI" use="required">
            <simpleType><restriction base="integer">
                <minInclusive value="0"/>
                <maxInclusive value="1"/>
            </restriction></simpleType>
        </attribute>
        <attribute name="FRR" use="optional">
            <simpleType><restriction base="epf:FrameRateType">
            </restriction></simpleType>
        </attribute>

        <attribute name="BRR" use="optional">
            <simpleType><restriction base="epf:FrameRateType">
            </restriction></simpleType>
        </attribute>

        <attribute name="FMT" use="optional">
            <simpleType><restriction base="epf:ListZeroOneType">
            </restriction></simpleType>
        </attribute>

        <attribute name="BMT" use="optional">
            <simpleType><restriction base="epf:ListZeroOneType">
            </restriction></simpleType>
        </attribute>

        <attribute name="FQU" use="optional">
            <simpleType><restriction base="integer">
                <minInclusive value="0"/>
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2006 – All rights reserved

```
            <maxInclusive value="31"/>
        </restriction></simpleType>
    </attribute>
    <attribute name="BQU" use="optional">
        <simpleType><restriction base="integer">
            <minInclusive value="0"/>
            <maxInclusive value="31"/>
        </restriction></simpleType>
    </attribute>
    <attribute name="FIX" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="68"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="FIN" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="68"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="FPX" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="68"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="FPN" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="68"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="BIX" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="296"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="BIN" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="296"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="BPX" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="296"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="BPN" use="optional">
        <simpleType>
            <restriction base="integer">
                <length value="296"/>
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="STC" use="optional">
        <simpleType><restriction base="epf:ListZeroOneType">
        </restriction></simpleType>
    </attribute>
    <attribute name="COD" use="optional">
        <simpleType><restriction base="epf:ListZeroOneType">
```

© ISO/IEC 2006 – All rights reserved

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

```
                </restriction></simpleType>
            </attribute>

        <attribute name="FTC" use="optional">
                <simpleType><restriction base="integer">
                    <minInclusive value="0"/>
                    <maxInclusive value="262143"/>
                </restriction></simpleType>
            </attribute>
            <attribute name="BTC" use="optional">
                <simpleType><restriction base="integer">
                    <minInclusive value="0"/>
                    <maxInclusive value="262143"/>
                </restriction></simpleType>
            </attribute>
            <attribute name="FSK" use="optional">
                <simpleType><restriction base="integer">
                    <minInclusive value="0"/>
                </restriction></simpleType>
            </attribute>
            <attribute name="BSK" use="optional">
                <simpleType><restriction base="integer">
                    <minInclusive value="0"/>
                </restriction></simpleType>
            </attribute>

            <attribute name="FGN" use="optional">
                <simpleType><restriction base="epf:ListZeroOneType">
                </restriction></simpleType>
            </attribute>

            <attribute name="BGN" use="optional">
                <simpleType><restriction base="epf:ListZeroOneType">
                </restriction></simpleType>
            </attribute>

        </complexType>
    </element>
</schema>
```

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

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

© ISO/IEC 2006 – All rights reserved

Not for Resale

Annex F
(normative)

Node coding tables

F.1 Node Coding tables

The coding tables for the nodes defined in this international standard are presented in electronic attachment to this document.

For the complete set of node coding tables (including all the scene graph nodes defined by MPEG-4) see the electronic attachement of ISO/IEC 14496-11.

F.2 Node Definition Type tables

The coding tables for the node definition types defined in this international standard are presented in electronic attachment to this document. For the complete set of node definition type tables (including all the scene graph nodes defined by MPEG-4) see the electronic attachement of ISO/IEC 14496-11.

© ISO/IEC 2006 – All rights reserved

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

Annex G
(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.

Company
IMEC
Samsung Electronics Co., Ltd.
Superscape

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] ISO/IEC 14772-1:1997, *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding*, http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[2] ISO/IEC 15444-1, *Information technology — JPEG 2000 image coding system — Part 1: Core coding system*

[3] ITU-T Rec. H.323:2000, *Packet-Based Multimedia Communications Systems*

[4] Virtual Reality Transfer Protocol, http://www.web3d.org/WorkingGroups/vrtp/

[5] "Scalar Vector Graphics (SVG) 1.0." W3C Recommendation. http://www.w3.org/TR/2000/CR-SVG-20000802/

[6] "Synchronized Multimedia Integration Language (SMIL) 2.0." W3C Recommendation. http://www.w3.org/AudioVideo/

[7] Living Worlds Proposal Draft 2, http://www.vrml.org/WorkingGroups/living-worlds/draft_2/index.htm

[8] Web3D Consortium, The Humanoid Animation Specification, http://www.h-anim.org

[9] Blaxxun Interactive. NURBS extension for VRML97. http://www.blaxxun.com/developer/contact/3d/nurbs/overview.html. April 1999.

[10] Intel 3D software toolkit, http://www.intel.com

[11] Abrash Michael. *Michael Abrash's graphics programming black book, special edition*. The Coriolis group, Inc.,Scottsdale, Arizona, 1997

[12] Abrash Michael. *Quake's lighting model: surface caching*. Dr. Dobb's sourcebook #260, November/December, 1996

[13] Barr A.H. *Global and Local deformations of solid primitives*. Computer Graphics, 18(3), 21-30, Proc. Siggraph '84

[14] Beeson Curtis. X3D Multitexture. http://www.web3d.org/TaskGroups/x3d/quadramix/multitexture.htm. May 1999.

[15] Biermann H., Levin A., Zorin D.. *Piecewise-smooth subdivision surfaces*. SIGGRAPH 2000 Conference Proceedings. New Orleans, Louisiana. July 23-28, 2000. pp. 113-120

[16] Bishop G., Oliveira M.M., *Relief Textures*. SIGGRAPH'2000 Proceedings

[17] Blinn J.F. *Simulation of wrinkled surfaces*. Computer Graphics, 123, pp. 286-292, 1978

[18] Blinn J.F. *A generalization of algebraic surface drawing*. ACM Transactions on Graphics, 1(3):235-256, July 1982.

[19] Bloomenthal J, *Polygonization of Implicit Surfaces*, Computer Aided Geometric Design, 5, 4 November 1988.

[20] Bloomenthal J., *Calculation of reference frames along a space curve*, Graphic Gems, Academic Press, 1990: pp. 567-571

[21] Briskin M., Elichai Y., Yomdin Y., *How can Singularity Theory help in Image Processing?* In "Pattern Formation in Biology, Vision and Dynamics", A. Carbone, M. Gromov and P. Prusinkiewitcz, Editors, World Scientific Publishers, 1999., pp. 392 – 423.

© ISO/IEC 2006 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[22] Catmull E. and Clark J. *Recursively generated B-spline surfaces on arbitrary topological meshes*. Computer-Aided Design, 10:350-355, September 1978.

[23] Chaikin G.M. *An algorithm for high-speed curve generation*. Computer Graphics Image Processing, 3:346-349, 1974.

[24] Chen S.E. *Quicktime VR - Image-Based Approach to Virtual Environment Navigation*, Proceedings of SIGGRAPH 95, Computer Graphics Proceedings, Annual Conference Series, pp. 29-38 (August 1995, Los Angeles, California). Addison Wesley. Edited by Robert Cook. ISBN 0-201-84776-0.

[25] Coquillard, Sabine. *Extended Free-Form Deformation: a sculpturing tool for 3D geometric modeling*. INRIA, RR-1250, June 1990.

[26] Craig John J. *Introduction to Robotics: Mechanics and Control, 2nd edition*. Addison-Wesley, Reading, MA, 1989.

[27] Debevec P., *Introduction to Image-Based Modeling, Rendering, and Lighting*. Siggraph'99 courses, 1999 (http://www.debevec.org/IBMR99/)

[28] Debevec P.E., Taylor C.J. and Malik J. *Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach*, Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series, pp. 11-20 (August 1996, New Orleans, Louisiana). Addison Wesley. Edited by Holly Rushmeier. ISBN 0-201-94800-1.

[29] Debevec P.E., Yu Y. and Borshukov G.D. *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*, Eurographics Rendering Workshop 1998, pp. 105-116 (June 1998, Vienna, Austria). Eurographics. Edited by George Drettakis and Nelson Max. ISBN 3-211-83213-0.

[30] DeRose T., Kass M. and Truong T. *Subdivision surfaces in character animation*. In SIGGRAPH'98 Conference Proceedings, ACM, Addison Wesley, pp. 85-94, July 1998.

[31] Diehl Stephen, *VRML++: a language for object-oriented virtual-reality models*, Proc. 24th International Conference on Technology of Object-Oriented Languages and Systems, Beijing, 1997

[32] Doo D. and Sabin M. *Behaviour of recursive division surfaces near extraordinary points*. Computer-Aided Design, 10(6):356-360, November 1978.

[33] Dyn N., Levin D. and Gregory J. A. *A four-point interpolatory subdivision scheme for curve design*, Computer-Aided Geometric Design, 4:257-268, 1987.

[34] Dyn N., Levin D. and Gregory J. A. *A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control*, ACM Transactions on Graphics, 9(2):160-169, April 1990.

[35] Dyn N., Levin D. and Liu D. *Interpolatory convexity-preserving subdivision schemes for curves and surfaces*, Computer-Aided Design, 24(4):211-216, April 1992.

[36] Eberly, D.H. *3D game engine design*, Academic Press, 2001

[37] Farin G., *Curves and surfaces for computer aided design, 2nd edition*, Academic Press, Boston 1990.

[38] Funge. J.D. *AI for Computer Games and Animation: A Cognitive Modeling Approach*. A K Peters Ltd, August 1999.

[39] Gersho A., Gray R. M. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[40] Guskov Igor, Vidimce Kiril, Sweldens Wim and Schröder Peter. *Normal Meshes*. Computer Graphics Proceedings (SIGGRAPH 2000), pp 95-102, 2000

[41] Hoppe Hugues, DeRose Tony, Duchamp Tom, Halstead Mark, Jin Huber, McDonald John, Schweitzer Jean, and Stuetzle Werner. *Piecewise smooth surface reconstruction*. In Computer Graphics Proceedings, Annual Conference Series, pages 295–302. ACM Siggraph, 1994.

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[42] Howard P. G. and Vitter J. S., *Arithmetic coding for data compression*, Proc. IEEE, vol. 82, no. 6, pp. 857-865, June 1994.

[43] Johnstone J.K. and Williams J.P. *A rational quaternion spline of arbitrary continuity*. Tech. Report.http://www.cis.uab.edu/info/faculty/jj/cos.html, 1999

[44] Kleene S.C. *Logique Mathématique*. Éditions Jacques Gabay, Paris, 1967.

[45] Kleene S.C. *Introduction to Metamathematics*. Wolters-Noordhoff Publishing, Groningen, 1971.

[46] Kobbelt L. $\sqrt{3}$ *-Subdivision*. SIGGRAPH'00 Conference Proceedings, pp. 103-112, July 2000.

[47] Kochanek D.H.U. *Interpolating splines with local tension, continuity, and bias control*. Computer Graphics, 18(3): 33-41, Proc. Siggraph '84.

[48] Khodakovsky A., Schröder P., and Sweldens W. *Progressive Geometry Compression*. SIGGRAPH 2000 proceedings

[49] Lamport Leslie, Time, *Clocks and the Ordering of Events in a Distributed System*, Communications of the ACM, 21(7), July 1978, pages 558-565, QA75.A68.

[50] Levoy M., Rusinkiewicz S., *QSplat: A Multiresolution Point Rendering system for Large Meshes*. SIGGRAPH'2000 Proceedings

[51] Loop C. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, August 1987.

[52] Lounsbery M., DeRose T., and Warren J. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. ACM Trans. on Graphics, 16(1), Jan. 1997.

[53] Luebke David and Georges Chris Department of Computer Science University of North Carolina at Chapel Hill. *Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets*. http://www.cs.virginia.edu/~luebke/publications/portals.html

[54] Mallat S., *A Theory for Multiresolution Signal Decomposition: The Wavelet Representation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 7, pp. 674-693, July 1989.

[55] Mignot A., Bourges-Sevenier M., Rotgé J.-F., *Solid Model Representation*, AFX proposal m7729, ISO/IEC/JTC1/SC29WG11 MPEG2001, Pattaya, December 2001

[56] Moller Tomas and Haines Eric, *Real-time rendering*. A K Peters Ltd., June 1999.

[57] Morán F. and García N. *Hierarchical Coding of 3D Models with Subdivision Surfaces*. IEEE ICIP 2000 proceedings.

[58] Munteanu A., Cornelis J., Van der Auwera G., and Cristea P., *A wavelet based lossless compression scheme with progressive transmission capability*, Int. J. of Imaging Systems and Technology, John Wiley&Sons, 10(1): 76-85, 1999.

[59] Munteanu A., Cornelis J., Van der Auwera G., and Cristea P., *Wavelet Image Compression - The Quadtree Coding Approach*, IEEE Transactions on Information Technology in Biomedicine, vol. 3, no. 3, pp. 176-185, September 1999.

[60] Nishimura H. and al. *Object modeling by distribution function and a method of image generation*. Trans. IECE Japan, Part D, J68-D(4):718-725, 1985.

[61] Nishino K., Sato Y. and Ikeuchi K. *Eigen-Texture Method: Appearance Compression based on 3D Model*. IEEE Computer Vision and Pattern Recognition, pp. I:618-624.

[62] Perlin K. and Hoffert E.M. *Hypertextures*. Conference proceedings on Computer graphics, 1989, pp. 253 -262

[63] Piegl and Tiller. *The NURBS book*. Springer-Verlag. 1997

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

© ISO/IEC 2006 – All rights reserved

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

[64] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. *Numerical Recipes in C*, Cambridge University Press, 2nd Edition, 1992.

[65] Roehl Bernie. *Regions*, http://ece.uwaterloo.ca/~broehl/vrml/regions.html

[66] Rotgé, J.-Fr. *L'arithmétique des formes: une introduction à la logique de l'espace*, PhD thesis, Université de Montréal, Montréal, 1997.

[67] Rotgé, J.-F., *Principles of solid geometry design logic*, p 233-254, in Proceedings of the CSG 96 Conference, Winchester, UK, 17-19 April 1996.

[68] Rotgé, J.-Fr. *Geometry Manual*. SGDL Systems Inc., Montreal, 2001.

[69] Said A. and Pearlman W.A.. Image compression using the spatial-orientation tree. Proc. IEEE Int. Symp. Circuits and Syst., Chicago, IL, pages 279-- 282, May 1993.

[70] Said A. and Pearlman W., *A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, no. 3, pp. 243-250, 1996.

[71] Salomie A., Deklerck R., Munteanu A., Cornelis J., *The MeshGrid Surface Representation*, Technical Report IRIS-TR-0082, Dept. ETRO-IRIS, Fac. TW, Vrije Universiteit Brussel, April 2002, http://www.etro.vub.ac.be/MeshGrid_TR_0082_2002.pdf.

[72] Samet H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[73] Schweitzer J. E. *Analysis and Application of Subdivision Surfaces*, PhD thesis, University of Washington, August 1996.

[74] Sederberg and Parry. *Free-Form Deformation of solid geometric models*. SIGGRAPH'96, vol 20, pp. 151-160. ACM, August 1996.

[75] SGDL Systems Inc, *Reference Manual*, SGDL Systems Inc., Montreal, 2001

[76] Shade J., Gortler S., Hey L., Szeliski R., *Layered Depth Images*. SIGGRAPH'97 Proceedings

[77] Shapiro J.M., *Embedded Image Coding Using Zerotrees of Wavelet Coefficients*, IEEE Transactions on Signal Processing, vol. 41, no. 12, pp. 3445-3462, 1993.

[78] Shoemake Ken, *Euler Angle Conversion*, Gramphics Gems IV", Academic Press Professional, Toronto, 1994

[79] Signoroni A. and Leonardi R., *Progressive ROI Coding and Diagnostic Quality for Medical Image Compression*, Proceedings of SPIE, Vol. 3309, pp. 674-685, 1997.

[80] Singhal S.K. and Cheridon D.R. *Exploiting position history for efficient remote rendering in networked VR*. In Presence: Teleoperators and Ves, 4(2):169-199.

[81] Schröder Peter and Sweldens Wim. *Spherical wavelets*: efficiently representing functions on the sphere. Computer Graphics Proceedings (SIGGRAPH 95), pp 161-172, 1995

[82] Ström J. and Cosman P. C., *Medical image compression with lossless regions of interest*, Signal Processing, vol. 59, no., pp. 155-171, 1997.

[83] Taubin G. A *Signal Processing Approach to Fair Surface Design*. SIGGRAPH'95 Conference Proceedings, pp. 351-358, August 1995.

[84] Vlachos A., Isidoro J. *Smooth C^2 Quaternion-Based Flythrough Paths*, Game Programming Gems 2, Charles River Media, 2001

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[85] Vlasic D., Pfister H., Molinov S., Grzeszczuk R. and Matusik W., *Opacity Light Fields: Interactive Rendering of Surface Light Fields with View-dependent Opacity*, Proc. Of 2003 symposium on Interactive 3D graphics, pp. 65-74, 2003.

[86] Wallin A., *Constructing isosurfaces from CT data*. In IEEE Computer Graphics and Applications, 11(6)28-33, November 1991.

[87] Warren J. *Subdivision Methods for Geometric Design*. Technical Report (110 pages), Rice University, November 1995.

[88] Watt A. and Watt M., *Advanced animation and rendering techniques*. ACM Press, 1992.

[89] Williams L. *Pyramidal parametrics*, Computer Graphics, 17(3):1-11, Proc. Siggraph '83.

[90] Wood D. N., Azuma D. I., Aldinger K., Curless B., Duchamp T., Salesin D. H., Stuetzle W. *Surface Light Fields for 3D Photography*, Proceedings of SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series, pp. 287-296 (July 2000). ACM Press / ACM SIGGRAPH / Addison Wesley Longman. Edited by Kurt Akeley. ISBN 1-58113-208-5.

[91] Wyvill Brian and Kees van Overveld. *Metamorphosis of Boolean Compound Soft Objects*. Implicit Surfaces '96, Eurographics, 1996.

[92] Yu T., Lin N., Liu S. J., and Chan A. K., *A Region-of-Interest Based Transmission Protocol for Wavelet-Compressed Medical Images*, Proceedings of SPIE Conference on Wavelet Applications IV, Vol. 3078, pp. 56-64, April 1997.

[93] Zonenschein L. et al., *Texturing Composite Deformable Implicit Objects*, Proceedings of SIBGRAPI '98, http://www.visgraf.impa.br/RefBib/Data/PS_PDF/texture-sib98/SLE085.pdf

[94] Zorin D., Schröder P. and Sweldens W. *Interpolating Subdivision for Meshes with Arbitrary Topology*, SIGGRAPH'96 Conference Proceedings, pp. 189-192, August 1996.

[95] Zorin D. *Subdivision and Multiresolution Surface Representations*, PhD thesis, California Institute of Technology, January 1998 (submitted in September 1997).

[96] Zorin D., Schröder P. et al. Subdivision for Modeling and Animation, SIGGRAPH'00 Conference Course Notes, July 2000.

[97] Zwicker, M., Pfister, H., VAN Baar, J., Gross, M. H. Surface Splatting. SIGGRAPH 2001 Conference Proceedings, [Los Angeles, USA]: SIGGRAPH, August 12-17, 2001, pp. 371-378 [cited 21 April 2005].

[98] Web3D Consortium - Open Standards for Real-Time 3D Communication, 21 April 2005, http://www.web3d.org.

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

© ISO/IEC 2006 – All rights reserved

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

ICS 35.040

Price based on 209 pages

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

© ISO/IEC 2006 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale