INTERNATIONAL STANDARD

ISO/IEC 14496-16

Third edition
2009-12-15

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 14496-16:2009(E)

Documents
Document Sharing Hub
© ISO/IEC 2009

Full document details: ISO_IEC 14496-16_2009

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.

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2009

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

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 14496-16_2009

Contents

Page

Foreword vi

1 Scope 1

2 Normative references 1

3 Symbols and abbreviated terms 1
3.1 Introduction 2
3.2 The AFX place within computer animation framework 3
3.3 Geometry tools 5
3.3.1 Non-Uniform Rational B-Spline (NURBS) 5
3.3.2 Subdivision surfaces 8
3.3.3 MeshGrid representation 28
3.3.4 MorphSpace 34
3.3.5 MultiResolution FootPrint-Based Representation 35
3.3.6 Solid representation 39
3.4 Texture tools 49
3.4.1 Depth Image-Based Representation 49
3.4.2 Depth Image-based Representation Version 2 53
3.4.3 Multitexturing 57
3.5 Animation tools 62
3.5.1 Deformation tools 62
3.5.2 Generic skeleton, muscle and skin-based model definition and animation 64
3.6 Rendering tools 74
3.6.1 Shadows 74

4 AFX bitstream specification - 3D Graphics compression tools 76
4.1 Introduction 76
4.2 Geometry tools 77
4.2.1 3DMC Extension 77
4.2.2 Wavelet Subdivision Surfaces 134
4.2.3 MeshGrid stream 137
4.2.4 MultiResolution FootPrint-Based Representation 174
4.3 Texture tools 182
4.3.1 Depth Image-Based Representation 182
4.3.2 PointTexture stream 187
4.4 Animation tools 197
4.4.1 Bone-based animation 197
4.4.2 Frame-based Animated Mesh Compression (FAMC) stream 211
4.5 Generic tools 234
4.5.1 Multiplexing of 3D Compression Streams: the MPEG-4 3D Graphics stream (.m3d) syntax 234
4.5.2 AFX Generic Backchannel 238

5 AFX object codes 249

6 3D Graphics Profiles 250
6.1 Introduction 250
6.2 "Graphics" Dimension 251
6.2.1 MPEG-4 X3D Interactive Graphics Profiles and Levels 251
6.2.2 MPEG-4 "Basic AFX" Graphics Profiles and Levels 254
6.3 "Scene Graph" Dimension 256
6.3.1 MPEG-4 X3D Interactive Scene Graph Profile and Levels 256
6.3.2 PEG-4 "Basic AFX" Scene Graph Profile and Levels 259
6.4 "3D Compression" Dimension 261
6.4.1 "Core 3D Compression" Profile and Levels 262

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: ISO_IEC 14496-16_2009

6.4.2 3D Multiresolution Compression Profile and Levels 265

7 XMT representation for AFX tools....269
7.1 AFX nodes 269
7.2 AFX encoding hints 269
7.2.1 WaveletSubdivision encoding hints 269
7.2.2 MeshGrid encoding hints....269
7.2.3 OctreeImage encoding hints 270
7.2.4 PointTexture encoding hints 270
7.2.5 BBA encoding hints....270
7.3 AFX encoding parameters 270
7.3.1 WaveletSubdivisionEncodingParameters....270
7.3.2 MeshGridEncodingParameters 271
7.3.3 PointTextureEncodingParameters....272
7.3.4 BBAEncodingParameters 272
7.4 AFX decoder specific info....273
7.4.1 WaveletSubdivision decoder specific info....273
7.4.2 MeshGrid decoder specific info 273
7.5 XMT for Bone-based Animation 274
7.5.1 BBA 274
7.5.2 BBA_header 274
7.5.3 BBA_encoding 275
7.5.4 BBA_body....275
7.5.5 BBA_frame 275
7.5.6 BBA_frameMask 275
7.5.7 BoneMask 276
7.5.8 MuscleMask....277
7.5.9 MorphMask 277
7.5.10 BBA_frameValues....278

Annex A (normative) Wavelet Mesh Decoding Process....279
A.1 Overview 279
A.2 Base mesh 279
A.3 Definitions and notations....279
A.4 Bitplanes extraction 280
A.5 Zero-tree decoder 281
A.6 Synthesis filters and mesh reconstruction....281
A.7 Basis change....282

Annex B (normative) MeshGrid Representation 283
B.1 The hierarchical multi-resolution MeshGrid 283
B.1.1 Building MeshGrid multi-resolution levels 283
B.1.2 Relation between the resolution level of the mesh and the number of reference-surfaces. 284
B.1.3 Relation between the resolution level of the grid and the number of reference-surfaces....284
B.1.4 Region Of Interest (ROI): Computation of the ROIs size and position....285
B.2 Scalability Modes....287
B.2.1 Scalability types....287
B.2.2 The mesh resolution scalability 288
B.2.3 The scalability in shape precision 288
B.2.4 The vertex position scalability 289
B.3 Animation Possibilities 290
B.3.1 Introduction 290
B.3.2 Vertex-based animation 290
B.3.3 Rippling effects, changing the offsets....291
B.3.4 Animation of the reference-grid points 291

Annex C (informative) MeshGrid representation 293
C.1 Representing Quadrilateral meshes in the MeshGrid format 293
C.2 IndexedFaceSet models represented by the MeshGrid format. 294
C.3 Computation of the number of ROIs at the highest resolution level given an optimal ROI size 294

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

Annex D (informative) **Solid representation** 296
D.1 Overview 296
D.2 Solid Primitives 296
D.3 Arithmetics of Forms 297
D.3.1 Introduction 297
D.3.2 General Arithmetic Operators on Densities 298
D.3.3 Ternary Logic: The Kleene Operators 300
D.3.4 Densities Filtering 302

Annex E (informative) **Face and Body animation: XMT compliant animation and encoding parameter file format** 303
E.1 XSD for FAP file format 303
E.2 XSD for BAP file format 304
E.3 XSD for EPF 305

Annex F (normative) **Local refinements for MultiResolution FootPrint-Based Representation** 309

Annex G (informative) **Partition Encoding for FAMC** 311

Annex H (informative) **Animation Weights Encoding for FAMC** 312

Annex I (normative) **Layered decomposition for FAMC** 313
I.1 Obtaining the layered decomposition 313
I.2 Deriving layers Id[l], l>0 using decoded data and connectivity (layeredDecompositionIsEncoded equals 1) 314
I.3 Deriving layers Id[l], l>0 using only connectivity (default option, layeredDecompositionIsEncoded equals 0) 314
I.4 Deriving layer Id[0] 317
I.5 Mesh simplification 319

Annex J (normative) **Reconstruction of values from decoded prediction errors with LD technique for FAMC** 321

Annex K (normative) **CABAC definitions, basic functions, and binarizations (as used for FAMC)** 324
K.1 CABAC definitions 324
K.1.1 Definition of the term CabacContext 324
K.1.2 Definition of the term `DecodingEnvironment` 324
K.2 CABAC basic functions 324
K.2.1 Definition of the function `cabac.biari_init_context( ctx, preCtxState )` 324
K.2.2 Definition of the function `cabac.arideco_start_decoding( dec_env )` 324
K.2.3 Definition of the function `cabac.biari_decode_symbol( dec_env, ctx )` 325
K.2.4 Definition of the function `cabac.biari_decode_symbol_eq_prob( dec_env )` 327
K.2.5 Definition of the function `cabac.biari_decode_final( dec_env )` 328
K.3 CABAC binarization functionss 328
K.3.1 Definition of the function `cabac.exp_golomb_decode_eq_prob( dec_env, k )` 328
K.3.2 Definition of the function `cabac.unary_exp_golomb_decode( dec_env, ctx, exp_start )` 328

Annex L (normative) **Node coding tables** 330
L.1 Node Coding tables 330
L.2 Node Definition Type tables 330

Annex M (informative) **Patent statements** 331

Bibliography 332

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

Foreword

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

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

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

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

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

This third edition cancels and replaces the second edition (ISO/IEC 14496-16:2006), of which it constitutes a minor revision. It also incorporates ISO/IEC 14496-16:2006/Amd.1:2007, ISO/IEC 14496-16:2006/Amd.1:2007/Cor.1:2008, ISO/IEC 14496-16:2006/Amd.1:2007/Cor.2:2008, ISO/IEC 14496-16:2006/Amd.2:2009, ISO/IEC 14496-16:2006/Amd.3:2008.

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*

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

— *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*

— *Part 19: Synthesized texture stream*

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

— *Part 21: MPEG-J Graphics Framework eXtensions (GFX)*

— *Part 22: Open Font Format*

— *Part 23: Symbolic Music Representation*

— *Part 24: Audio and systems interaction [Technical Report]*

— *Part 25: 3D Graphics Compression Model*

— *Part 27: 3D Graphics conformance*

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

Documents
Document Sharing Hub

Full document details: ISO_IEC 14496-16_2009

Information technology — Coding of audio-visual objects —

Part 16:
Animation Framework eXtension (AFX)

1 Scope

This Part of ISO/IEC 14496 specifies MPEG-4 Animation Framework eXtension (AFX) model for representing and encoding 3D graphics assets to be used standalone or integrated in interactive multimedia presentations (the latter when combined with other parts of MPEG-4). 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*

ISO/IEC 14496-25:2008, *Information technology — Coding of audio-visual objects — Part 25: 3D Graphics Compression Model*

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

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

OD Object Descriptor

VRML Virtual Reality Modelling Language

Animation Framework eXtension – 3D Graphics Primitives

3.1 Introduction

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)'s main objective is to propose compression scheme for static and animated 3D assets. This encoded version is encapsulated in an Elementary Stream, a similar approach as the one used by audio or video in MPEG-4. The AFX compression is closely connected to the definition of the graphics primitives in the scene graph. For such reason, AFX second objective is to update the scene graph when necessary and define new BIFS nodes for graphics primitives. However, AFX compression can also operate on other scene graph formalisms (as indicated in ISO/IEC 14496-25 a.k.a. MPEG-4 Part 25).

The place of AFX within MPEG-4 terminal architecture is illustrated in Figure 1.

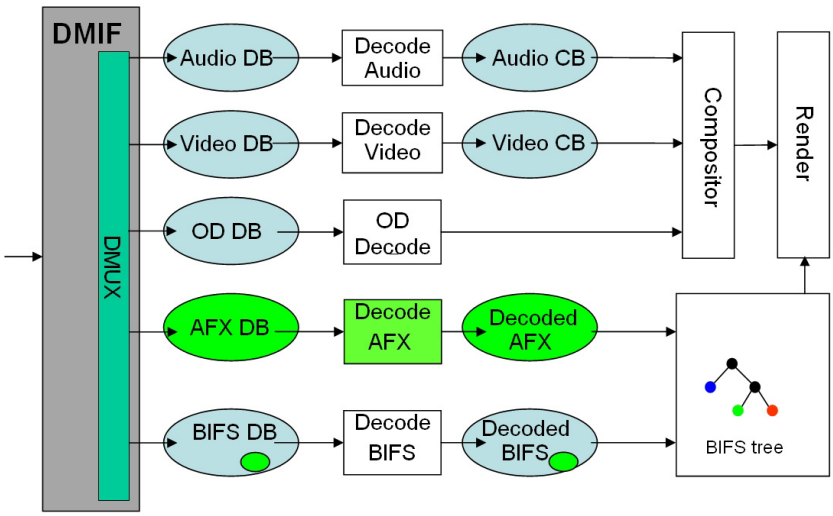


Figure 1 — Animation Framework eXtension (in green) and MPEG-4 Systems.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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

3.2 The AFX place within computer animation framework

To understand the place of AFX within computer animation framework [35], 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, the 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.

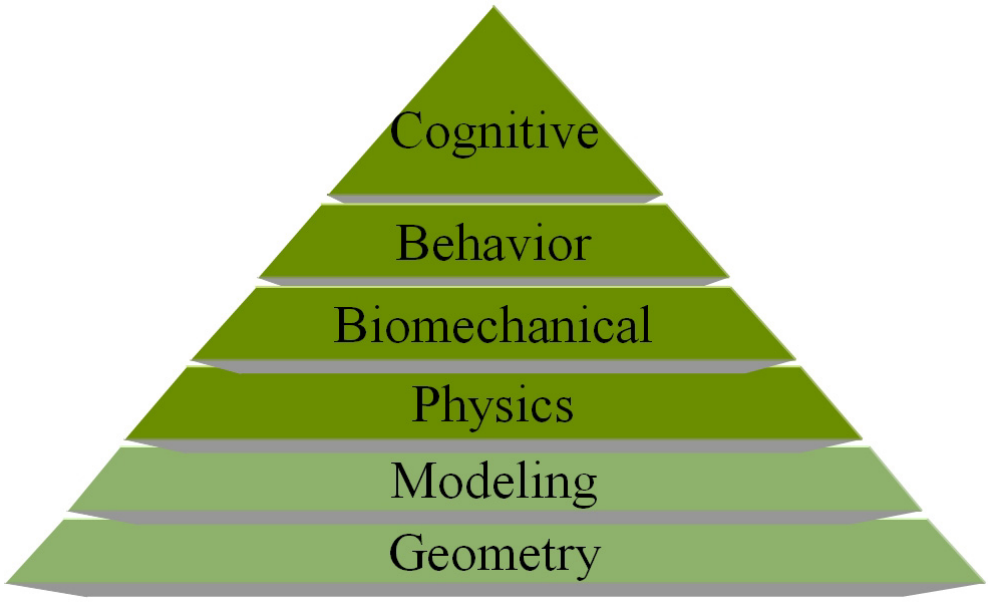


Figure 2 — Models in computer games and animation [35].

1) 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.

2) 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.

3) 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.

4) 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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

5) 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.

6) 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 two categories. Models of the last two categories are typically application-specific and often designed programmatically. 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 and Modeling components.

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 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. 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 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 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

3.3 Geometry tools

3.3.1 Non-Uniform Rational B-Spline (NURBS)

3.3.1.1 Introduction

A Non-Uniform Rational B-Spline (NURBS) curve of degree $p > 0$ (and hence of order p+1) and control points $\{\mathbf{P}_i\}$ ($0 \le i \le n-1$; $n \ge p+1$) is mathematically defined as [34], [60], [85]:

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

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-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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

- Local control: if the control point $\mathbf{P}_i$ is moved or the weight $\mathbf{w}_i$ 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:

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

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.

3.3.1.2 NurbsCurve

3.3.1.2.1 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]

}

3.3.1.2.2 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.

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.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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

3.3.1.3 NurbsCurve2D

3.3.1.3.1 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]

}

3.3.1.3.2 Functionality and semantics

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

3.3.1.4 NurbsSurface

3.3.1.4.1 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, ∞)	
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]

}

3.3.1.4.2 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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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: $0 \le i <$ **uDimension** $\ge$ **uOrder** and $0 \le j <$ **vDimension** $\ge$ **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* equals zero 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 on the control points defined in **controlPoint**, just like these fields apply on **coord** points of an **IndexedFaceSet**.

3.3.2 Subdivision surfaces

3.3.2.1 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.

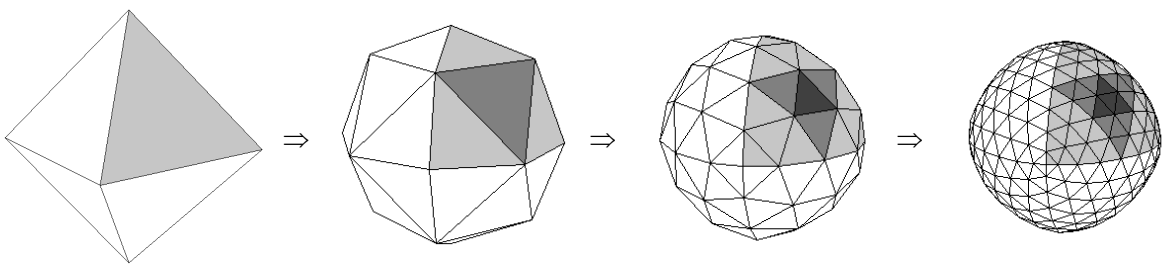


Figure 3 — Nested control meshes

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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.

3.3.2.1.1 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_3 and Q_1.

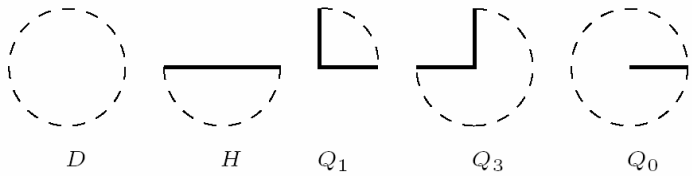


Figure 4 — The charts for a surface with piecewise smooth boundary

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.

Two adjacent smooth segments of a boundary are allowed to be joined, producing a crease ending in a *dart* (cf. [37]). For dart vertices an additional chart Q_0 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.

3.3.2.1.2 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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

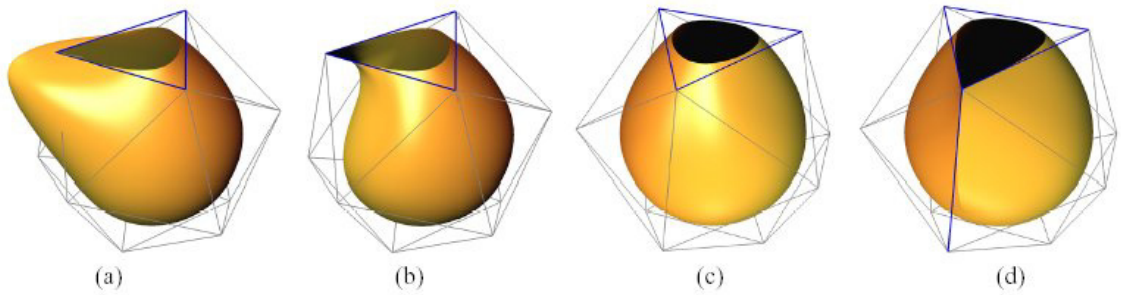


Figure 5 — Features: (a) concave corner, (b) convex corner, (c) smooth boundary/crease, (d) corner with two convex sectors.

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.

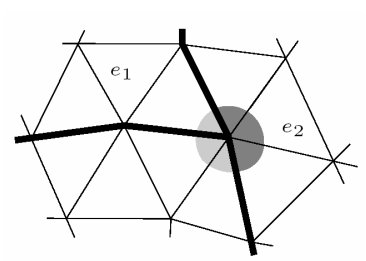


Figure 6 — Crease edges meeting in a corner with two convex (light grey) and one concave (dark grey) sectors. The subdivision scheme modifies the rules for edges incident to crease vertices (e.g. e_1) and corners (e.g. e_2).

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

3.3.2.2 SubdivisionSurface

3.3.2.2.1 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  MFSubdivSurfaceSectorNod   sectors               []
                  e
}
```

3.3.2.2.2 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 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 3.3.2.2.3.

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:

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

- 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:

1) Midpoint (non-smoothing);
2) Loop (subclause 3.3.2.2.3.1);
3) Dyn's butterfly (cf. [31], [91]);
4) Catmull-Clark (subclause 3.3.2.2.3.3).

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 3.3.2.2.3.2) 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 3.3.2.1.2). 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 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 3.3.2.1.2) 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.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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 3.3.2.2.3 for more details. Note that if both the **sector** and the **invisibleEdgeIndex** fields are non-empty, the results are undefined.

3.3.2.2.3 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.

3.3.2.2.3.1 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 [48].

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 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

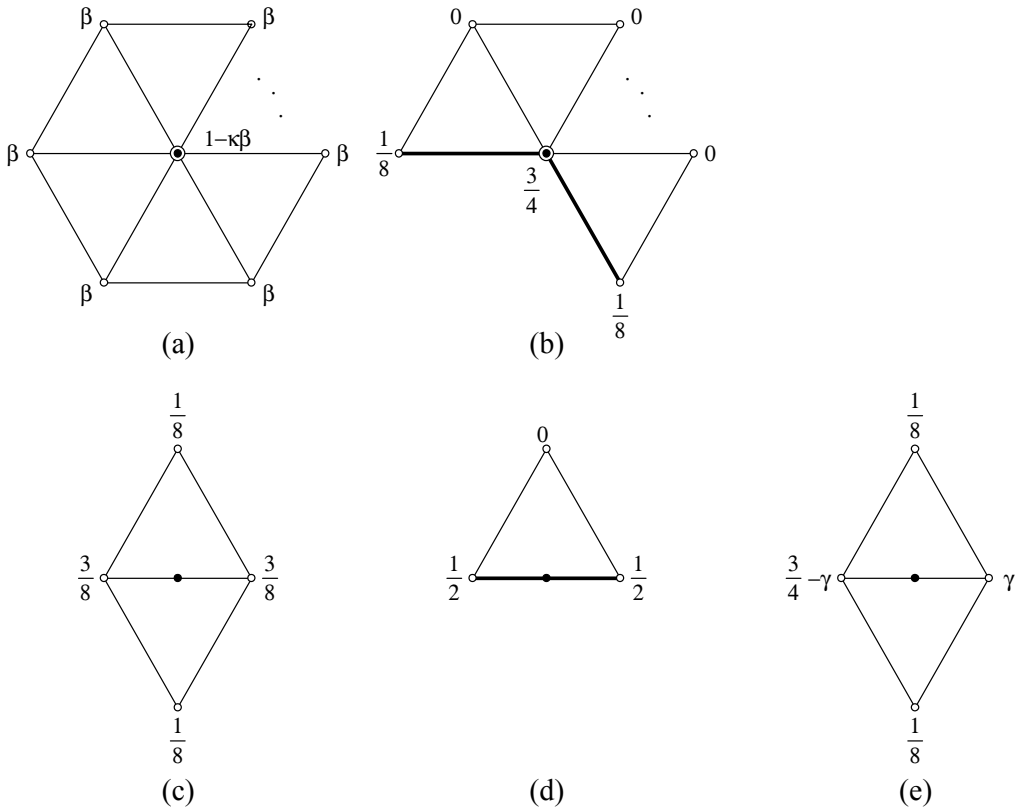


Figure 7 — Stencils of Loop's subdivision scheme. (a) repositioning interior old vertices, (b) repositioning boundary or crease old vertices, (c) splitting interior edges not adjacent to boundary extraordinary vertices, (d) splitting boundary and crease edges, (e) splitting interior edges adjacent to 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 β = 3/16 for *k* = 3, and β = 1/16 for *k* = 6. Warren proposed later to keep β = 3/16 for *k* = 3, but to have simply β = $3/8k$ for *k* > 3, which also amounts to β = 1/16 for *k* = 6 [84].

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:

- In the absence of tags, the standard edge rules are applied (Figure 7c).
- If a crease edge e is split, and no endpoint is a dart vertex, the rule shown in Figure 7d is used.
- If one of the endpoints is a dart vertex, the standard rule (Figure 7c) is applied.
- 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 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

- 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*, the edge *e* is differentiated whether it is in a convex or concave sector. For a convex corner, $\theta_k = \alpha / k$, where α 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.

- **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. $\mathbf{a}_i = (l^i, p)$, 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.

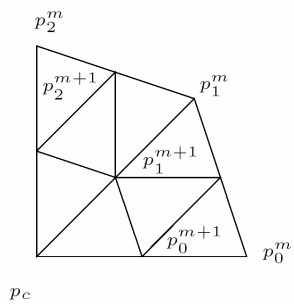


Figure 8 — Neighborhoods of a vertex on different subdivision levels. The subdivision matrix relates the vector of control points p^m to the control points on the next level p^{m+1}. For a neighborhood of k triangles $p^m = \{p_c^m, p_0^m, \ldots, p_{k-1}^m\}$.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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.

- **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* denotes 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 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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 $\theta = k\theta_k$.

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

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

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

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

3.3.2.2.3.2 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.

- **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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

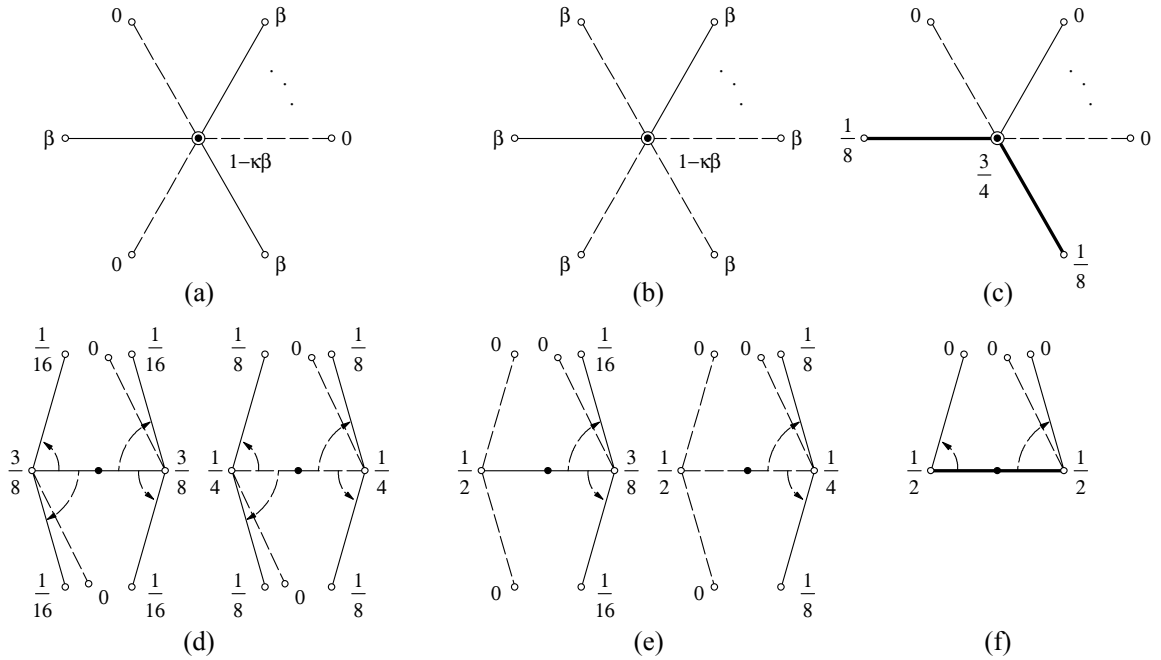


Figure 9 — Stencils of extended Loop subdivision for repositioning interior(smooth and dart) (a), (b) and boundary/crease old vertices and splitting interior (d), (e) and boundary/crease edges.

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.

- **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.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

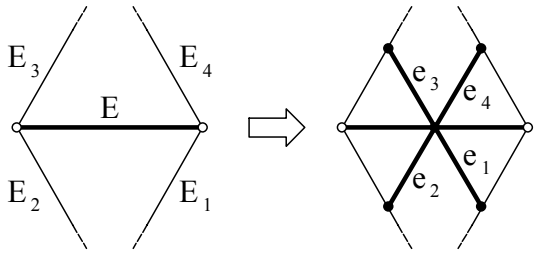


Figure 10 — New edges created when splitting an edge in two.

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 e1, e2, e3 and e4, these connect to other new points created by dividing the original edges labeled E1, E2, E3 and E4. The visibility of the new edges is initialized to invisible and is changed to visible if the lookup of the combination of E, E1, E2, E3 and E4 in Table 1 indicates that it should be. Clearly, the visibility of each of e1, e2, e3 and e4 is not uniquely determined by splitting E alone; e1, for example, also depends on the visibility lookup for splitting E1.

Table 1 — New edge visibility lookup table

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

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

- **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:

$$n = t_1 \times t_2 .$$

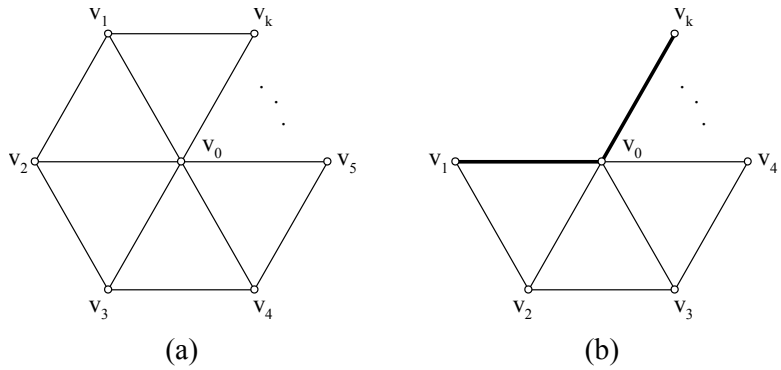


Figure 11 — (a) interior, and (b) boundary/crease vertex neighborhood used for normal calculation.

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 $i \in [1, k]$):

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

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

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

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

3.3.2.2.3.3 Catmull-Clark subdivision

The Catmull-Clark scheme was described in [19]. 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.

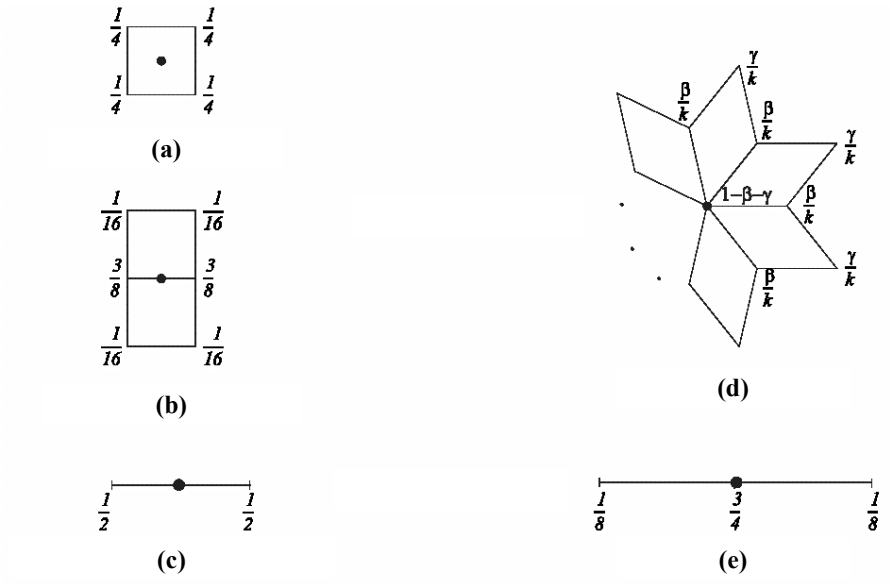


Figure 12 — Catmull-Clark rules with k defined as in subclause 3.3.2.1.2 (a) face rule; (b) regular edge rule (c) boundary/crease edge rule (d) interior vertex rule $\beta = 3/2k$ and $\delta = 1/4k$ (e) smooth boundary/crease vertex rule

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.

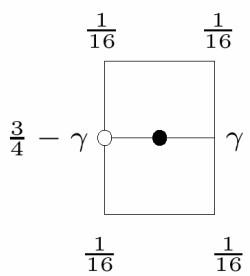


Figure 13 — Modified rule for odd vertices adjacent to a boundary extraordinary vertex (Catmull-Clark scheme), the coefficient gamma is defined in the text.

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 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

- 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.

- **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 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

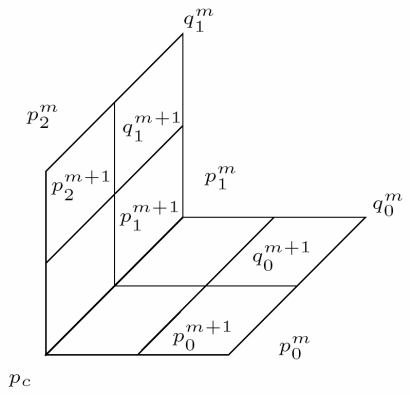


Figure 14 — Neighborhoods of a vertex on different subdivision levels. The subdivision matrix relates the vector of control points p^m to the control points on the next level p^{m+1}. For a neighborhood of k quadrilaterals $p^m = \{p_c^m, p_0^m, \ldots, p_{k-1}^m, q_0^m, \ldots, q_{k-1}^m\}$.

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. $\mathbf{a}_i = (l^i, p), l^0, l^1, l^2$ 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

$$\lambda = \frac{1}{2}\gamma + \frac{1}{8}c^2 + \frac{1}{16}\sqrt{64\gamma^2 + 32c^2\gamma - 48\gamma + 4c^4 + 4c^2 + 9}$$

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.

- **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 $\theta_k = 2\pi / k$.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

$$l_c^0 = \frac{k}{k+5}, \quad l_{pi}^0 = \frac{4}{k(k+5)}, \quad l_{qi}^0 = \frac{1}{k(k+5)}$$
$$x_c^1 = x_c^2 = l_c^1 = l_c^2 = 0$$
$$x_{pi}^1 = \frac{1}{\sigma}\sin i\theta_k \quad x_{pi}^2 = \frac{1}{\sigma}\cos i\theta_k$$
$$x_{qi}^1 = \frac{\sin i\theta_k + \sin(i+1)\theta_k}{\sigma(4\lambda-1)}, \quad x_{qi}^2 = \frac{\cos i\theta_k + \cos(i+1)\theta_k}{\sigma(4\lambda-1)},$$
$$l_{pi}^1 = 4\sin i\theta_k, \quad l_{pi}^2 = 4\cos i\theta_k,$$
$$l_{qi}^1 = \frac{\sin i\theta_k + \sin(i+1)\theta_k}{4\lambda-1}, \quad l_{qi}^2 = \frac{\cos i\theta_k + \cos(i+1)\theta_k}{4\lambda-1}$$

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 $k = 1$,

$$\begin{array}{llll} x_c^1 = 1/18 & x_{p0}^1 = -2/18 & x_{p1}^1 = -2/18 & x_{q0}^1 = -5/18 \\ x_c^2 = 0 & x_{p0}^2 = -1/2 & x_{p1}^2 = 1/2 & x_{q0}^2 = 0 \\ l_c^1 = 6 & l_{p0}^1 = -3 & l_{p1}^1 = -3 & l_{q0}^1 = 0 \\ l_c^2 = 0 & l_{p0}^2 = -1 & l_{p1}^2 = 1 & l_{q0}^2 = 0 \end{array}$$

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

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 lc, lp0 and lpk.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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

3.3.2.2.3.4 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 $a_i' = a_i - (a_i, n)n$ for $i = 1,2$.

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

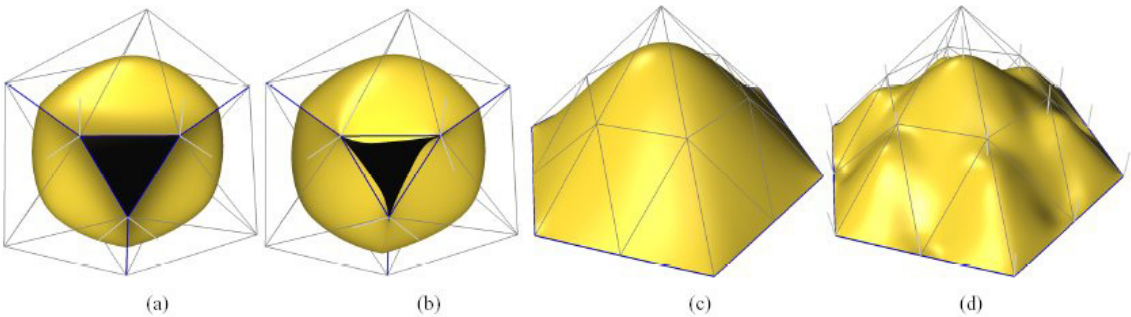


Figure 15 — Normal interpolation. (a) Surface with convex corners. (b) Prescribed directions: at each corner we tilt the normal for one surface sector slightly inwards. (c) Smooth surface. (d) Same control mesh but all normals vertical.

3.3.2.3 SubdivSurfaceSector

3.3.2.3.1 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
}
```

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

3.3.2.3.2 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.

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

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

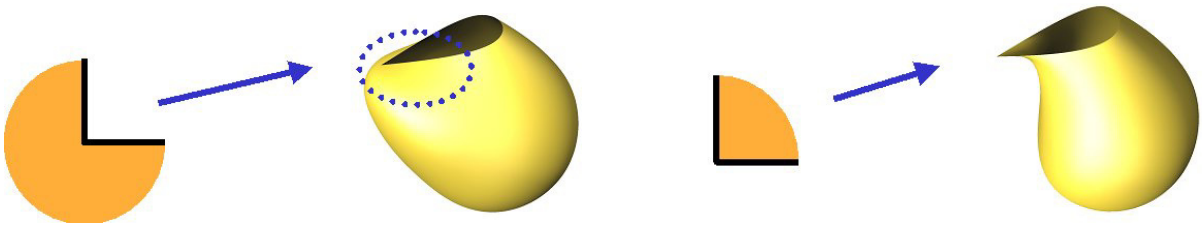


Figure 16 — Concave corners (left) and convex corners (right).

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).

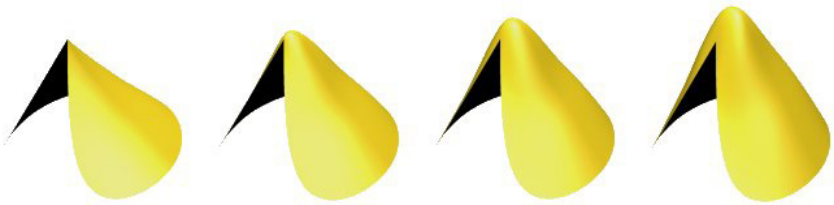


Figure 17 — Influence of the flatness parameter.

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).

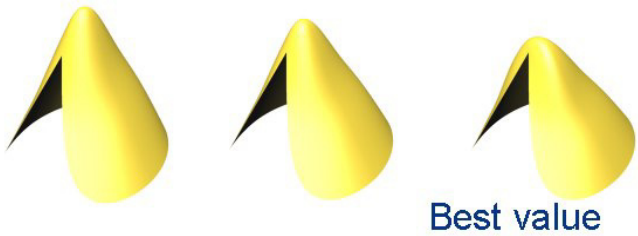


Figure 18 — Influence of theta.

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

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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).

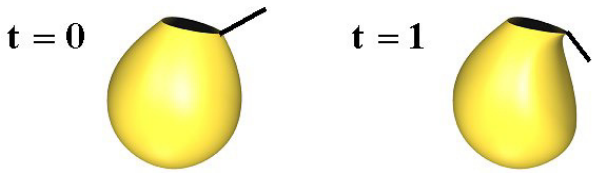


Figure 19 — Prescribed normals and tension parameter.

3.3.2.4 WaveletSubdivisionSurface

3.3.2.4.1 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
}
```

3.3.2.4.2 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 4.3.1.1).

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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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

3.3.3 MeshGrid representation

3.3.3.1 Introduction

The peculiarity of the MeshGrid representation [68] 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-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.

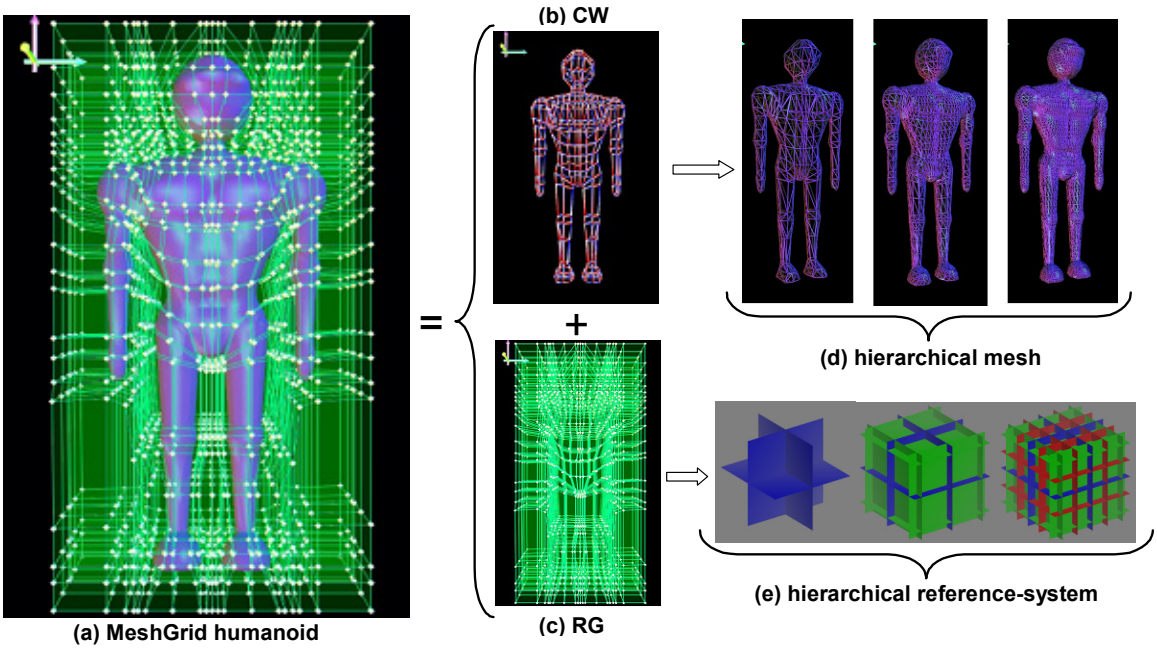


Figure 20 — A MeshGrid humanoid model designed by means of implicit surfaces.

3.3.3.1.1 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 following equation

$$RG = \bigcap \left\{ \sum_U S_U, \sum_V S_V, \sum_W S_W \right\}$$

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).

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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).

3.3.3.1.2 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 following equation

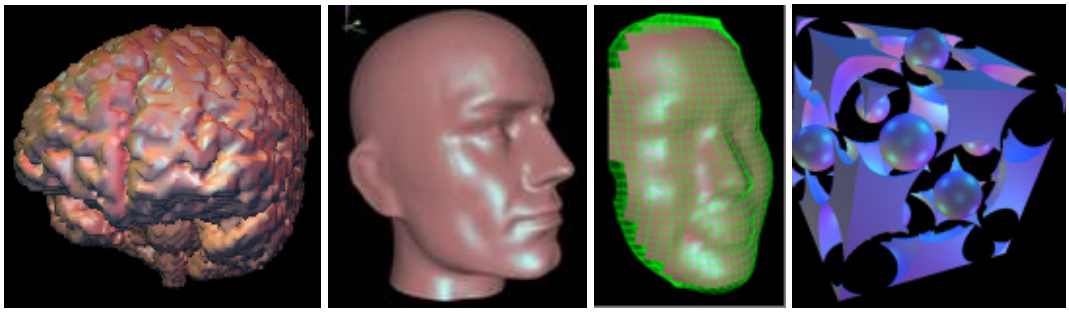
$$CW = \bigcup \left\{ \sum_U C(S_U), \sum_V C(S_V), \sum_W C(S_W) \right\}$$

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 3.3.3.1.1, results in a multi-resolution connectivity-wireframe, in which each resolution level consists of a single seamless mesh, as shown in Figure 20(d).

3.3.3.1.3 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 4.2.3) and particular animation possibilities, such as: hierarchical grid animation, and offset based animation (see subclause 3.3.3.3).

3.3.3.2 MeshGrid

3.3.3.2.1 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

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

exposedField	SFInt32	displayLevel	0
exposedField	SFInt32	filterType	0
exposedField	SFCoordinateNode	gridCoord	NULL
exposedField	SFInt32	hierarchicalLevel	0
exposedField	MFInt32	nLevels	[]
exposedField	SFNormalNode	normal	NULL
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	

}

3.3.3.2.2 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 4.2.3.2.36): **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 4.2.3.3.1 and 4.2.3.3.3.

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 4.2.3.3.4 and B.1.2.

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.1.2.

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

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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 4. 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 3.3.3.3.1.

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 4.2.3.3.1. 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 3.3.3.3.2, while the decoding of this field is described in subclause 4.2.3.3.3. 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 4.2.3.3.1). 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 4.2.3.3.1.

The **coordIndex** field is filled with the indices of the triangles when building the triangulation of the decoded mesh as explained in 4.2.3.3.1.3. 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 4.2.3.3.1).

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 4.2.3.3.1.

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 4.2.3.3.1)
01	Use smooth filter (see subclause 4.2.3.3.1)
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.1.2.

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

3.3.3.3 Animation Extensions

3.3.3.3.1 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 3.3.3.2.1 and 3.3.3.2.2), 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

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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" [30]. The position of a reference-grid point P_{2n}^{l+1} (shown in Figure 22) is computed as follows:

$$P_{2n}^{l+1} = \Delta P_{2n}^{l+1} + (-w \cdot P_{2n-3}^{l} + (0.5+w) \cdot P_{2n-1}^{l} + (0.5+w) \cdot P_{2n+1}^{l} - w \cdot P_{2n+3}^{l})$$

$$\Delta P_{2n}^{l+1} = P_{2n}^{l+1} + (-w \cdot P_{2n-3}^{l} + (0.5+w) \cdot P_{2n-1}^{l} + (0.5+w) \cdot P_{2n+1}^{l} - w \cdot P_{2n+3}^{l})$$

where l represents the hierarchical level of the grid point, P the last position of a point, P the new position, ΔP the detail computed for the last position P. The detail ΔP 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.

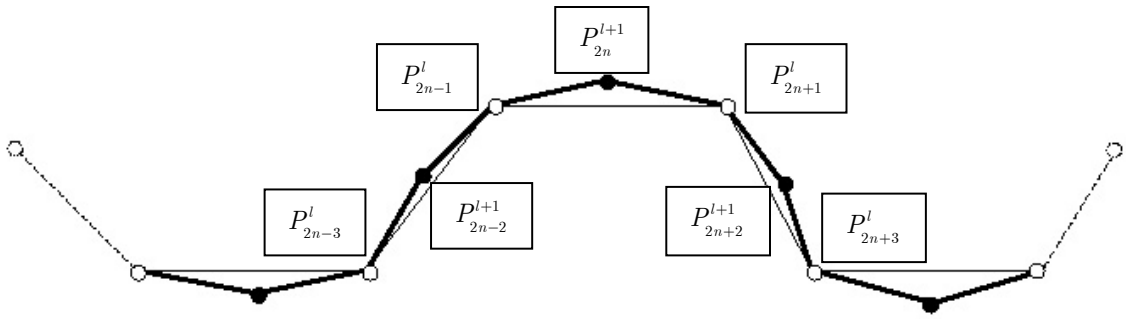


Figure 22 — Dyn's four point scheme for curves applied for the hierarchical MeshGrid.

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 above.

The position of the vertices is updated from the grid positions as explained in subclause 3.3.3.3.2.

3.3.3.3.2 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 3.3.3.1.1). 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 G1 to which it is attached, and one outside the object G2. The relative position of vertex V with respect to G1 and G2 is given by the scalar offset (label 3) expressed by $offset = \left|\overrightarrow{G_1V}\right| \Big/ \left|\overrightarrow{G_1G_2}\right|$ with

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

$offset \in [0,1)$. Because offset indicates the fractional position between G1 and G2, any displacement of G1 or G2 (see the example of Figure 23(b)) is automatically reflected in an update of the coordinates (x,y,z) of V: $\overrightarrow{OV} = \overrightarrow{OG_1} + \overrightarrow{G_1G_2} \cdot offset$.

- 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 $offset = \left|\overrightarrow{G_1V}\right| / offAmp + 0.5$ with $offset \in [0,1)$, where *offAmp* is the maximum amplitude of the offset defined by the *offsetAmplitude* variable specified in MeshGridDecoderConfig (subclause 4.2.3.2.2). 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: $\overrightarrow{OV} = \overrightarrow{OG_1} + \vec{N}(offAmp(offset - 0.5))$ where $\vec{N}$ is the normal vector to the surface at vertex *V*. In this case the displacements along the normal follow both directions as shown for the vertices with labels 4 and 5.

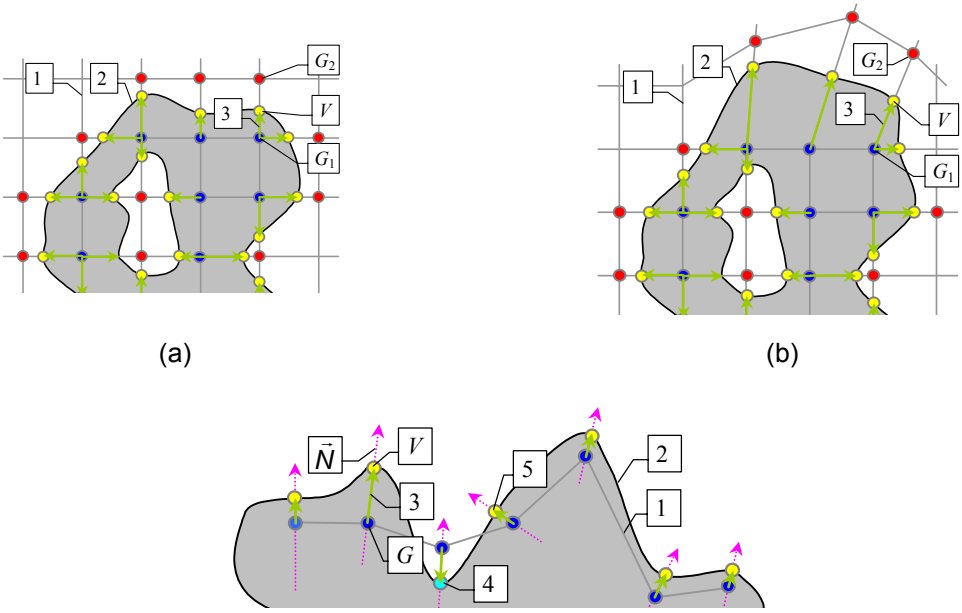


Figure 23 — A cross-section through a 3D object, illustrating the contour of the object, the reference-grid, and the relation between the vertices (belonging to the connectivity-wireframe and located at the surface of the object) and the grid points.

NOTE: In equations above O denotes the origin of the reference coordinate system in which the reference-grid is defined.

The offsets of the vertices are stored in the ***vertexOffset*** field of the **MeshGrid** node (see subclause 3.3.3.2).

3.3.3.4 Examples

The following example loads a VRML MeshGrid node.

```
Shape {
   appearance Appearance {
   material Material {
         diffuseColor 1.0 0.0 0.0
         ambientIntensity 0.1
```

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

```
        }
    }
    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,
           ...
          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
      ]
    }
}
```

3.3.4 MorphSpace

3.3.4.1 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.

3.3.4.2 MorphShape node

3.3.4.2.1 Syntax

MorphShape{ #%NDT=SF3DNode,SF2DNode

exposedField	SFInt32	morphID	
exposedField	SFShapeNode	baseShape	
exposedField	MFShapeNode	targetShapes	[ ]
exposedField	MFFloat	weights	[ ]

}

3.3.4.2.2 Semantics

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

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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).

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

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

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 above 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*.

3.3.5 MultiResolution FootPrint-Based Representation

3.3.5.1 Introduction

MultiResolution FootPrint-Based representation is a solution to represent any set of objects based on footprints (a set of IndexedLineSet, or for a near future, buildings, cartoons…). The main interests in this representation are its progressivity, view dependency, and compression.

3.3.5.2 FootPrintSetNode

3.3.5.2.1 Node Interface

```
FootPrintSetNode { #NDT=%SFGeometryNode
    exposedField    MFGeometryNode          children            []
}
```

3.3.5.2.2 Functionality and semantics

The **children** field specifies the list of all footprints rendered according to the current viewpoint. This list contains currently FootPrintNode representing the set of footprints rendered from the current viewpoint. This list can be updated at each displacement of the viewpoint in order to adapt the scene complexity to the view. This representation can be extended to be used with any object based on footprints such as buildings, cartoons, etc. In this case, the children field can contain BuildingPartNode to represent buildings.

3.3.5.3 FootPrintNode

3.3.5.3.1 Node Interface

```
FootPrintNode {  #NDT=%SFGeometryNode
    exposedField    SFInteger               index               -1
```

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

```
    exposedField    SFIndexLineSet2D        footprint           NULL
}
```

3.3.5.3.2 Functionality and semantics

Index: this is the index of the node corresponding to a footprint elevation at a specific level of detail. This index is essential for streaming, due to the synchronization between the representation on the server and on the client. This index will be sent to the server as a refinement request.

Footprint: this is an IndexLineSet2D describing the footprint.

3.3.5.4 BuildingPartPrintNode

3.3.5.4.1 Node Interface

```
BuildingPartNode { #NDT=%SFGeometryNode
    exposedField    SFInteger               index               -1
    exposedField    SFIndexLineSet2D        footprint           NULL
    exposedField    SFUnsigned integer      buildingIndex       -1
    exposedField    SFFloat                 height              0
    exposedField    SFFloat                 altitude            0
    exposedField    MFGeometryNode          alternativeGeometry []
    exposedField    MFRoofNode              roofs               []
    exposedField    MFFacadeNode            facades             []
}
```

3.3.5.4.2 Functionality and semantics

Index: this is the index of the node corresponding to a footprint elevation at a specific level of detail. This index is essential for streaming, due to the synchronization between the representation on the server and on the client (this index will be sent to the server as a refinement request).

Footprint: this is a IndexLineSet2D describing the footprint.

buildingIndex: this is the index of the building to which this part is connected. A building corresponds to a group of building parts having the same buildingIndex.

Height: this is the height of the building.

Altitude: this is the altitude of the building (corresponding to the floor of the prism).

alternativeGeometry: this is a geometry node corresponding to an optional object used to replace the normal building. This alternative geometry can be used to swap a building with a more detailed model (used for example to replace a footprint elevation based model of a monument, by a more detailed model). In this case, the footprint-based elevation model will not be rendered, since the alternative model will be.

roofs: this is a node array allowing to describe complete roofs that will be reconstructed on top of the footprint elevation.

facades: this is a node array allowing to describe in detail the modelling of the façades corresponding to this building part. The size of this array corresponds to the number of facades, equivalent to the number of edges of the polygon defining the footprint.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

3.3.5.5 RoofNode

3.3.5.5.1 Node Interface

```
RoofNode { #NDT=%SFGeometryNode
    exposedField    SFInteger       type                0
    exposedField    SFFloat         height              0.0
    exposedField    MFFloat         slopeAngle          [0.0]
    exposedField    SFFloat         eaveProjection      0.0
    exposedField    SFInt           edgeSupportIndex    -1
    exposedField    SFURL           roofTextureURL      ""
    exposedField    SFBool          isGenericTexture    TRUE
    exposedField    SFFloat         textureXScale       1.0
    exposedField    SFFloat         textureYScale       1.0
    exposedField    SFFloat         textureXPosition    0.0
    exposedField    SFFloat         textureYPosition    0.0
    exposedField    SFFloat         textureRotation     0.0
}
```

3.3.5.5.2 Functionality and semantics

type: this is the type of the roof. 0 – Flat Roof, 1 – Symmetric Hip Roof, 2 – Gable Roof, 3 – Salt Box roof, 4 – Non Symmetric Hip Roof.

height: this is the height of the roof that allows cropping it. (This is not used for flat roofs).

slopeAngle: this is the angle of the roof slopes in degrees (useless for flat roofs). In the case of a Symmetric Hip Roof, all slopes have the same angle. In the case of a Non Symmetric Hip Roof, each slope has a specific angle.

eaveProjection: this is the projection of the eave (useless for flat roofs).

edgeSupportIndex: this is the index of the edge in the footprint that supports the roof (use only for Salt Box roofs)

roofTextureURL: this is the URL of the texture that is orthogonally mapped onto the roof

isGenericTexture: this specifies whether the texture mapped onto the roof is generic or not. In the case of a generic texture, the reference system is centred on the top left vertex of the roof pan, and axed perpendicularly to the gutter. In the case of an aerial photograph, the reference system is centred on the first vertex of the footprint, and axed on the world coordinate system.

textureXScale: this is the scaling of the roof texture along X-axis

textureYScale: this is the scaling of the roof texture along Y-axis

textureXPosition: this is the displacement of the texture along X-axis

textureYPosition: this is the displacement of the texture along Y-axis

textureRotation: this is an angle in radian specifying the rotation to apply to the texture.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

3.3.5.6 FacadeNode

3.3.5.6.1 Node Interface

FacadeNode { #NDT=%SFGeometryNode

exposedField	SFFloat	WidthRatio	1.0
exposedField	SFFloat	XScale	1.0
exposedField	SFFloat	YScale	1.0
exposedField	SFFloat	XPosition	0.0
exposedField	SFFloat	YPosition	0.0
exposedField	SFFloat	XRepeatInterval	0.0
exposedField	SFFloat	YRepeatInterval	0.0
exposedField	SFBool	Repeat	FALSE
exposedField	SFURL	FacadePrimitive	""
exposedField	SFInteger	NbStories	0
exposedField	MFInteger	NbFacadeCellsByStorey	0
exposedField	MFFloat	StoreyHeight	1.0
exposedField	MFFacadeNode	FacadeCellsArray	[]

}

3.3.5.6.2 Functionality and semantics

WidthRatio: this corresponds to a ratio between the width of the cell compared to the width of the parent cells.

XScale: this is a parameter allowing scaling in X-coordinate the model corresponding to the URL Façade Primitive (2D texture or 3D model). For texture, this scale corresponds to the real size in X-coordinate in meters of the texture. For 3D Model, this size corresponds to the scale to apply on the model in X-coordinate.

NOTE: This scale is very important as the model can be used for different buildings, and must be adjusted to the current one.

YScale is a parameter allowing scaling in Y-coordinate the model (2D texture or 3D model). For texture, this scale corresponds to the real size in Y-coordinate in meters of the texture. For 3D Model, this size corresponds to the scale to apply on the model in Y-coordinate.

NOTE: This scale is very important as the model can be used for different buildings, and must be adjusted to the current one.

XPosition: this is a parameter allowing moving in X-coordinate the model in the cell defined by the FacadePrimitiveArray of the father node. This position can be essential to place a primitive in the centre of the cell.

YPosition: this is a parameter allowing moving in Y-coordinate the model in the cell defined by the FacadePrimitiveArray of the father node. This position can be essential to place a primitive in the center of the cell.

Repeat: this is a Boolean that is TRUE if and only if the model has to be repeated all over the cell defined by the father node.

NOTE: This is essential for texture mapping, or to regularly repeat a model of windows all over a façade.

FacadePrimitive: this is a link to the corresponding primitive (Texture or 3D model) that have to be mapped onto the cell.

NbStories: this is the number of stories of the façade.

NbFacadeCellsByStorey: this is an array that defines the number of cells by storey. This parameter is essential to know on which storey corresponds a cell in FaçadeCellsArray.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

StoriesHeight: this is an array specifying the height of each storey.

FacadeCellsArray: this is an array of FacadeNode that links each cell to a facadeNode (another array of cells, and/or a façade primitive like a texture or a 3D model). The size of this array is the sum of all NbFacadeCellsByStorey[i] , for all i from 0 to NbStories.

3.3.6 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.

3.3.6.1 Solid modeling nodes

3.3.6.1.1 Implicit

3.3.6.1.1.1 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)

}

3.3.6.1.1.2 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 a 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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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

3.3.6.1.2 Quadric

3.3.6.1.2.1 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)
}
```

3.3.6.1.2.2 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 Figure 24) 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.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

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 3.3.6.6.1.1.2).

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 3.3.6.1.3). The density value is encoded as a 32-bit integer.

- **Quadric's Construction Mechanism**

Based on the principle of projective geometry, a geometrical construction mechanism for quadrics has been defined in [63], [65], [52] 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.

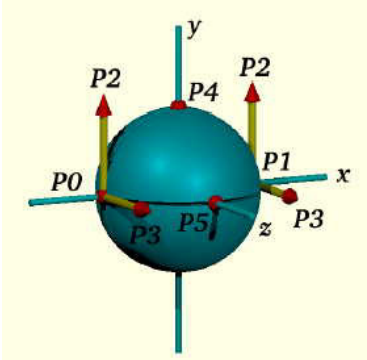


Figure 24 — Quadric's 6 geometric control points

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.

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009

It is possible to determine the 10 coefficients of the quadric's general equation from the previously described quadric's geometrical construction system. (Reference: [63] – pages 100 – 114).

- **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 (X, Y, Z). 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 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) also noted (2: 5: 0: 6).

3.3.6.1.3 SolidRep

3.3.6.1.3.1 Node interface

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

3.3.6.1.3.2 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 [90] is an example of an algorithm well suited for implicit surfaces.

3.3.6.2 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.

© ISO/IEC 2009 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 14496-16_2009