
**Information technology — Coding of
audio-visual objects —**

Part 16:

Animation Framework eXtension (AFX)

AMENDMENT 1: Geometry and shadow

Technologies de l'information — Codage des objets audiovisuels —

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

AMENDEMENT 1: Géométrie et ombre

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 2007

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

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.

Amendment 1 to ISO/IEC 14496-16:2006 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Coding of audio-visual objects —

Part 16:

Animation Framework eXtension (AFX)

AMENDMENT 1: Geometry and shadow

Add subclause 4.3.6 MultiResolution FootPrint-Based Representation:

4.3.6 MultiResolution FootPrint-Based Representation

4.3.6.1 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.

4.3.6.2 FootPrintSetNode

4.3.6.2.1 Node Interface

FootPrintSetNode { #NDT=%SFGGeometryNode

exposedField	MFGeometryNode	children	[]
}			

4.3.6.2.2 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.

4.3.6.3 FootPrintNode

4.3.6.3.1 Node Interface

FootPrintNode { #NDT=%SFGGeometryNode

exposedField	SFInteger	index	-1
exposedField	SFIndexLineSet2D	footprint	NULL
}			

4.3.6.3.2 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.

4.3.6.4 BuildingPartPrintNode

4.3.6.4.1 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	[]

}

4.3.6.4.2 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.

4.3.6.5 RoofNode

4.3.6.5.1 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
}			

4.3.6.5.2 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.

4.3.6.6 FacadeNode

4.3.6.6.1 Node Interface

FacadeNode { #NDT=%SFGGeometryNode
}

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	[]

4.3.6.6.2 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 repeat regularly 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.

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.

Add subclause 4.7.3 BBA Animation Algorithm:

4.7.3 BBA Animation algorithm

The initial pose of an articulated model must contain a skeleton that is aligned with the mesh. Thus some bones have a non-identity initial transformation. During the animation, the bone transforms are updated. Since the skeleton and the mesh are originally aligned, only the offset between the new bone transforms and the initial ones has to be applied to the vertices.

Animating the skinned model consists then in the following steps:

- a) for all bones compute the initial transformation in the local space as the combination of the elementary transform: rotation, translation, center, scale and scaleOrientation; all these components are expressed in the parent's coordinate system.
- b) for all the bones, compute the initial transformation in the world space as a product between the initial transformation of the bone in the local space and the initial transformation of the bone's parent expressed in the world space
- c) compute the inverse of the previous transformation
- d) at each animation frame, update the local elementary transforms: rotation, translation, center, scale and scaleOrientation
- e) at each animation frame, repeat step b)
- f) at each animation frame, for all the bones multiply the transformation obtained at step e) with the one computed at step c)
- g) at each animation frame, for each pair bone/vertex, multiply the vertex with the transform obtained at the previous step and with the corresponding weight.

Add Subclause 4.8 Scene tools:

4.8 Scene tools

4.8.1 Shadows

The **Shadow** node works as a special grouping node for the author defined creation of hard and soft shadows caused by 3D-surfaces, shadow properties and **SpotLight** nodes.

4.8.1.1 Syntax

```
Shadow {
    eventIn      MFNode      addChilden
    eventIn      MFNode      removeChildren
    exposedField MFNode      children
    exposedField SFFBool     enabled
    exposedField MFFBool     cast
                                []
                                TRUE
                                TRUE
```

```

    exposedField    MFBool    receive    TRUE
    exposedField    SFFloat    penumbra    0
}

```

4.8.1.2 Semantics

addChildren: the addChildren event appends nodes to the grouping node's children field. Any nodes passed to the addChildren event that are already in the group's children list are ignored.

removeChildren: the removeChildren event removes nodes from the grouping node's children field. Any nodes in the removeChildren event that are not in the grouping node's children list are ignored.

children: contains a list of children nodes. The **children** node's surfaces are generally invisibly rendered. Only instances of 3D-surfaces are able to work as occluders and receivers for shadow creation in association only with **SpotLight** nodes. Each **children[m]** and its descendants correspond to the combination of shadow properties **cast[m]** and **receive[m]**. If it is intended, that a 3D-surface has to work as occluder or receiver, it must fulfil several prerequisites. The assigned children has to be a single instance 3D-surface or an instance 3D-surface that is part of a sub-graph. A **SpotLight** must be associated to this 3D-surface in the same way as shown in Figure AMD1-2. The light source has to illuminate the 3D-surface, for casting or receiving shadows.

enabled: the functionality of the **Shadow** node is enabled with the value TRUE. The functionality of the **Shadow** node is disabled with the value FALSE.

cast: assigns the capability to a 3D-surface to cast shadows onto other 3D-surfaces. With the value TRUE a single instance or a branch with instances of 3D-surfaces included becomes an occluder. The field works as **MFBool**, so every **children[m]** (single node or branch) is able to have its own value of **cast**. The shadow properties of a 3D-surface's node instance are transmitted according the ID of MediaObject to all of those instances of 3D-surfaces existing outside of **Shadow** nodes in that scene with the same ID (see Figure AMD1-2).

receive: assigns the capability to a 3D-surface to receive shadows from itself or from other surfaces. With TRUE a single instance or a branch with instances of 3D-surfaces included becomes a receiver. The field works as **MFBool**, so every **children[m]** (single node or branch) is able to have its own value of **receive**. The shadow properties of a 3D-surface's node instance are transmitted according the ID of MediaObject to all of those instances of 3D-surfaces existing outside of **Shadow** nodes in that scene with the same ID (see Figure AMD1-2). The field works as **MFBool**, so every **children[m]** node is able to have its own value of **receive**. The shadow properties of a 3D-surface's node instance become transmitted to all of those instances of 3D-surfaces existing outside of **Shadow** nodes in that scene (see Figure AMD1-3).

penumbra: describes the geometrical extension of the related **SpotLight** as a sphere radius.

4.8.1.3 Annex

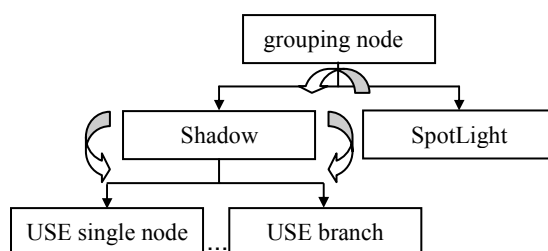


Figure AMD1-1 — Semantical representation

If **Shadow** and **SpotLight** nodes own the same grouping node as parent, a shadow relationship is created automatically between them.

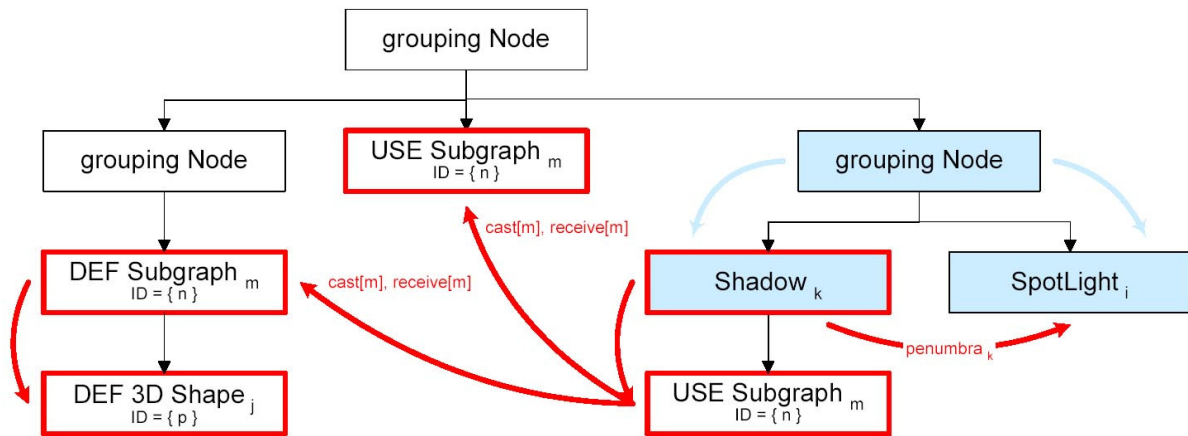


Figure AMD1-2 — Transmission of shadow properties to 3D-surfaces and light sources

The combination of multiple **Shadow** nodes with one or multiple **SpotLight** nodes possesses several shadow properties associated with one **SpotLight** node. This way a **SpotLight** node gets several **penumbra** values (see Figure AMD1-3). Additionally it can create multiple shadow properties with a single **Shape** node.

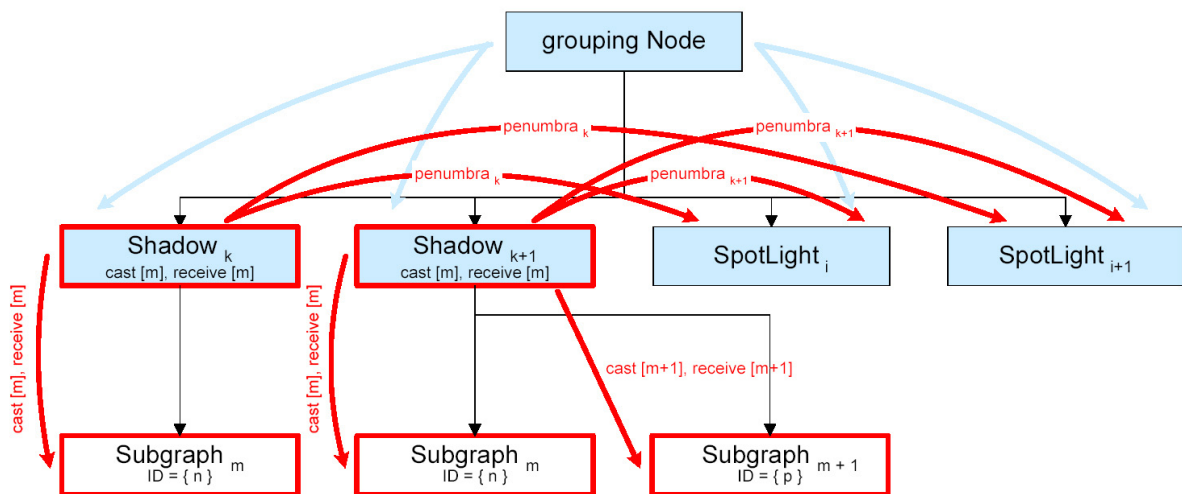


Figure AMD1-3 — Transmission of shadow properties to multiple 3D-surfaces and light sources

The following rules were formulated for those cases. If a unique 3D-surface is related to the same light source several times, the shadow properties are handled by the Boolean operation OR. All **penumbra** values add up and increase the light body extension.

The occurrence of multiple associated **SpotLight** nodes combined with only one **Shadow** node simply results in each **SpotLight** node creating another independent relation.

Add subclause 5.8 MultiResolution FootPrint-Based Representation:

5.8 MultiResolution FootPrint-Based Representation

5.8.1 Downstream syntax

This is the syntax of the downstream MultiResolution FootPrint-based Representation.

5.8.1.1 FootPrintSetDecoderSpecificInfo

5.8.1.1.1 Syntax

```
class FootPrintsDecoderConfig extends AFXDecoderSpecificInfo {
    SDLInt<16>    FPObjType
    SDLInt<32>    MaxNbFootPrints
    SDLInt<6>     FootPrintNbBits
    SDLFloat     Step
    SDLInt<6>     NbBitsMetricError
    SDLFloat     MinX
    SDLFloat     MaxX
    SDLFloat     MinY
    SDLFloat     MaxY
    SDLInt<1>     DEFIDUsed
    switch (FPObjType)
    {
        case 1 :    FPBuildingDecoderConfig FPBuildingDSI
    }
}
```

5.8.1.1.2 Semantics

FPObjType: This is an integer specifying the type of the multiresolution footprint-based representation (0 for classic footprints, but extended types could be considered.)

MaxNbFootPrints: this is the number of footprints in the footprint-based representation.

FootPrintNbBits: this is the number of bits used to decode the footprint indices. Its value is the lowest integer superior or equal to $\log_2(\text{MaxNbFootprints})$.

Step: this is the smallest spatial subdivision.

NbBitsMetricError: this is the number of bits on which to encode the metric error (32 or 64 bits)

MinX: this is the minimum X-coordinate of the model

MaxX: this is the maximum X-coordinate of the model

MinY: this is the minimum Y-coordinate of the model

MaxY: this is the maximum Y-coordinate of the model

DEFIDUsed: If **DEFIDUsed** is **TRUE**, then the ID used during the Bifs encoding is used as reference. Otherwise, the string defname is used.

NOTE For future extension, depending of the object type (buildings, cartoons...) other parameters could be added to this decoder configuration. For current simple footprints, these parameters are enough to configure the decoder.

5.8.1.2 FPBuildingDecoderSpecificInfo

5.8.1.2.1 Syntax

```
class FPBuildingDecoderConfig {
    SDLFloat    MinAltitude;
    SDLFloat    MaxHeight;
    SDLInt<6>    NbBitsZBuilding;
    SDLInt<6>    NbBitsNbStories;
    SDLInt<6>    NbBitsStoreyHeight;
    SDLInt<6>    NbBitsFacadeWidth;
    SDLInt<6>    NbBitsNbFacadeCellsByStorey;
}
```

5.8.1.2.2 Semantics

MinAltitude: this is the minimum altitude of the set of footprint-based elevations.

MaxHeight: this is the maximum height of the set of footprint-based elevations.

NbBitsZBuilding: this is the number of bits used to encode the altitude and height of buildings.

NbBitsNbStories: this is the number of bits used to specify the number of stories by façade element

NbBitsStoreyHeight: this is the number of bits used to specify the height ratio of each storey.

NbBitsFacadeWidth: this is the number of bits used to specify the width ratio of each façade element.

NbBitsNbFacadeCellsByStorey: this is the number of bits used to specify the number of cells per storey.

5.8.1.3 FootPrintSet Message

The FootPrintSet Message is intended to carry all the set base and refinement information for the design of footprint sets.

5.8.1.3.1 Syntax

```
FootPrintSetMessage {
    int(32) NbFootPrints
    For (int i=0; i<NbFootprints; i++)
        FootPrintMessage    FootPrint;
}
```

5.8.1.3.2 semantics

NbFootprints: this is an integer giving the number of FootPrintMessage that have to be read in the stream.

5.8.1.4 FootPrint Message

A Footprint message is intended to carry a base or refinement information for the design of footprint sets.

5.8.1.4.1 Syntax

```
class FootprintMessage {
    int(FPNbBits) index
    bit(1) type
    FPNewVertices FPNV
    int(6) IndexNbBits
    if (type) {
```

```

int(10) offspring
for (i=0; i<offspring; i++) {
    int(FPNbBits) localIndex
    float(NbBitsMetricError) MetricError
    IndexFootprintSet IFPS
    switch (FPObjectType)
    {
        case 1: FPBuildingParameters FPBP
    }
}
else
{
    float(NbBitsMetricError) MetricError
    int(8) NbRings
    For (int i=0; i<NbRings-1; i++)
    {
        int(IndexNbBits) FirstVertexIndex
        switch (FPObjectType)
        {
            case 1: FPBuildingParameters FPBP
        }
    }
}
}

```

5.8.1.4.2 Semantics

MetricError: this is the geometric error between the original model and the simplified model used by the client to decide if this node has to be refined.

IndexNbBits: this is the number of bits used to decode the vertices indexes. Its value is the lowest integer superior or equal to $\log_2(\text{FootPrintsDecoderConfig.MaxIndex})$.

Index: this is the index identifying the current footprint.

Type: this is a Boolean with value 0 if the current message describes a primary footprint, and 1 if this is a refinement.

FPNV: this is a class describing the new vertices used to refine the current footprint.

Offspring: this is the number of children of the current footprint.

localIndex: this is the index identifying the i-th child of the current footprint.

IFPS: this is a class listing the indices of vertices of the footprint.

NbRings: this is the number of rings in the new footprint.

FirstVertexIndex: this is the index in the new vertices array of the first vertex for each ring (there is no index for the first ring, since it is always equal to 0).

FPBP: this is a class describing the parameters corresponding to the new building based on footprint.

5.8.1.5 FPNewVertices

5.8.1.5.1 Syntax

```

class FPNewVertices
{
    int(6) coordtype

```

```

int(16) nbNewVertices
for (i=0; i<nbnewvertices; i++)
{
    if (type == 0 || step == -1.0)
    {
        float(32)    DeltaX
        float(32)    DeltaY
    }
    else
    {
        bool          SignDeltaX
        unsigned int(coordtype-1) AbsDeltaX
        bool          SignDeltaY
        unsigned int(coordtype-1) AbsdeltaY
    }
}
}

```

5.8.1.5.2 Semantics

coordType: this is the number of bits to encode the vertex coordinates.

nbNewVertices: this is the number of vertices described in the rest of the class.

DeltaX, DeltaY: these are the 2D coordinates of the newly added vertex,

SignDeltaX, SignDeltaY: these specify whether deltaX and deltaY are positive or not.

AbsdeltaX, AbsdeltaY: these are the 2D absolute value coordinates of the newly added vertex, expressed in a reference system based on the barycentre of the parent footprint vertices. The actual position of the new vertex is obtained by multiplying **AbsDeltax*SignDeltaX** by **Step** (defined in the DecoderSpecificInfo), and adding the coordinates of the barycentre of the parent footprint vertices.

The decoding process is exposed in Annex J.

5.8.1.6 IndexFootprintSet

5.8.1.6.1 Syntax

```

class IndexFootprintSet
{
    int(16)    nbVertexIndices
    for (int i=0; i<nbVertexIndices; i++)
    {
        int(IndexNbBits) index
    }
}

```

5.8.1.6.2 Semantics

nbVertexIndices: this is the number of indices in the rest of the class. **nbVertexIndices=nbVertices** in the footprint + Number of rings-1.

index: this is the index of the i-th vertex. If index=-1, a new ring starts.

5.8.1.7 FPBuildingParameters

5.8.1.7.1 Syntax

```

class FPBuildingParameters
{
    int(FPNbBits) buildingIndex
    if (step!=-1.0)
    {
        int(NbBitsZBuilding) altitude
        int(NbBitsZBuilding) height
    }
    else
    {
        float(32) altitude
        float(32) height
    }
    For (int i=0; i<nbFacades; i++)
    {
        FPFacade facades
    }
    int(6) nbRoofs
    for (int i=0; i<nbRoofs; i++)
    {
        FPRoof roof
    }

    int(8) nbSwapNodes
    for (int i=0; i<NbSwapNodes; i++)
    {
        if (DEFIDUse)
            int(32) nodeId
        else
        {
            int(8) nbChar
            for (int j=0; j<nbChar; j++)
                char defname[j]
        }
    }
}

```

5.8.1.7.2 Semantics

buildingIndex: this is the index of the building to which this part is connected.

altitude: if **step** is different from -1.0, the altitude of the building is encoded using a integer. The actual altitude is given by

$$\text{Decoded} = \text{altitude} * \text{Step}.$$

height: if **step** is different from -1.0, the height of the building is encoded using a integer. The actual height is given by:

$$\text{DecodedHeight} = \text{height} * \text{Step}.$$

textureURL: this is the URL of the texture to be applied on the side of the footprint-based elevation.

nbSwapNodes: this is the number of nodes in the scene graph used to swap the block corresponding to the footprint-based elevation.

NodeId: this is the index of a node in the scene graph used to swap the block corresponding to the current footprint-based elevation for a more defined model.

NOTE This footprint-based model swap for a more detailed model is essential if one needs to add a good detailed model that can be represent using a footprint-based representation. For example, one might need to replace the footprint-based model of specific buildings like monuments (used as reference points for the user during the navigation) by a more detailed model that can be representing by a footprint-based elevation.

nbRoofs: this is the number of roofs that are superimposed on top of the footprint elevation.

roof: this is the description of each roof that are superimposed on top of the footprint elevation.

nbfacades: this is the number of facades of the building, equivalent to the number of edges of the polygon defining by the footprint, i.e. equal to nbVertexIndices.

facades: this is the description of each facade model that are mapped on each face of the footprint elevation.

5.8.1.8 RoofBitStream

5.8.1.8.1 Syntax

```
class FPRoof {
    int(3)  roofType
    switch (roofType)
    {
        case 0 : //Flat Roof

        case 1 : // Symmetric Hip Roof
            float roofHeight
            float roofSlopeAngle
            float roofEaveProjection
        case 2 : // Gable Roof
            float roofHeight
            float roofSlopeAngle
            float roofEaveProjection
        case 3 : // Salt Box Roof
            float roofHeight
            float roofSlopeAngle
            float roofEaveProjection
            int(indexNbBits) roofEdgeSupportIndex
        case 4 : // Non Symmetric Hip Roof
            float roofHeight
            float roofSlopeAngle[nbWalls]
            float roofEaveProjection
    }
    if (DEFIDUse)
        int(32) nodeIdAppearance
    else
    {
        int(8) nbChar
        for (int j=0; j<nbChar; j++)
            char defnameAppearance [j]
    }

    bool  IsGeneric
    int(2) projectionTextureMode
    switch (projectionTextureMode) {
        case 0 : // Generic metric texture

        case 1 : // Generic texture
            float XScale
            float YScale
    }
}
```

```

    case 2 : // Real texture
        float XScale
        float YScale
        float XPosition
        float YPosition
        float rotation
        float roofEaveProjection
    }
}

```

5.8.1.8.2 Semantics

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

roofHeight: this is the height of the roof. If -1.0, the height is not defined; else, the roof is cropped.

roofSlopeAngle: this is the angle of the roof slopes.

roofEaveProjection: this is the length of the roof eave projection

roofEdgeSupportIndex: this is the index of the edge of the polygon defining the footprint elevation that supports the roof.

nodeIDAppearance or defnameAppearance: this is a reference to a Appearance node corresponding to the texture that is orthogonally mapped on the roof. **nodeIDAppearance** is used if DEFIDUsed is true (see FootprintDSI) else **defnameAppearance**

IsGeneric this specifies whether the texture mapped on the roof is generic (value 1), or obtained from an aerial photograph (value 0). If the roof texture is generic, the reference system is centred on the bottom left vertex of each roof pan, and aligned on the gutter. In the case of a non-generic texture, the reference system is centred on the first vertex of the footprint, and aligned along the world coordinate system.

projectionTextureMode: this is the mode used to map the texture on the roof. It specifies whether the different parameters XScale, YScale, XPosition, YPosition, and rotation are used.

XScale: this is the scaling the roof texture along X-axis

YScale: this is the scaling the roof texture along Y-axis

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

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

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

5.8.1.9 FacadeBitStream

5.8.1.9.1 Syntax

```

class FPFacade
{
    int (NbBitsFacadeWidth) WidthRatio
    int (j2) MappingMode
    Switch (MappingMode)
    {
        Case 0:

```

```

    Case 1:
        float(32) XScale
        float(32) YScale
    Case 2:
        float(32) XScale
        float(32) YScale
        float(32) XPosition
        float(32) YPosition
    Case 3:
        float(32) XScale
        float(32) YScale
        float(32) XPosition
        float(32) YPosition
        float(32) XRepeatInterval
        float(32) YRepeatInterval
}
Bool Repeat
int(2) FacadePrimitiveType
if (DEFIDUse)
    int(32) nodeIdFacadePrimitiveNode
else
{
    int(8) nbChar
    for (int j=0; j<nbChar; j++)
        char defnameFacadePrimitiveNode [j]
}

int(NbBitsNbStories) NbStories
For (int i=0; i<NbStories; i++)
{
    float(NbBitsStoreyHeight) StoreyHeight
}
For (int i=0; i<NbStories; i++)
{
    int(NbBitsNbFacadeCellsByStorey) NbFacadeCellsByStorey
    For (int j=0; j< NbFacadeCellsByStorey; j++)
    {
        FPFacade FacadeCell
    }
}
}

```

5.8.1.9.2 Semantics

XScale: this is the scale on X-coordinate applied over the model defined by FacadePrimitiveModel

YScale: this is the scale on Y-coordinate applied over the model defined by FacadePrimitiveModel

XPosition: this is the translation on X-coordinate applied over the model defined by FacadePrimitiveModel

YPosition: this is the translation on Y-coordinate applied over the model defined by FacadePrimitiveModel

Repeat: if this field has value 1 the texture or model on the façade is repeated.

FacadePrimitiveType: specifies the type of primitive:

0: nothing

1: texture

2: 3Dmodel.

nodeIdFacadePrimitiveNode or **defnameFacadePrimitiveNode**: this is a reference to the node in the scene graph corresponding to the Facade primitive. **nodeIdFacadePrimitiveNode** is used if **DEFIDUsed** is true (see **FootprintDSI**) else **defnameFacadePrimitiveNode**

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

NbFacadeCellsByStorey: this is the number of cells for the corresponding storey.

StoreyHeight is the height of the corresponding storey.

NbFacadeCells: this is the number of cells on the façade. It's equal to the sum of **NbFacadeCellsByStorey** for each storey.

StoreyWidth: this is the width in meter of the corresponding cell.

FacadeCell: this is a façade node corresponding to the cell.

5.8.2 Upstream syntax (for backchannel)

When specified as an upstream in corresponding ES descriptor, the **MultiResolution Footprint Set** stream has to be read according to the **AFX Generic Backchannel syntax** (see subclause 5.5).

Add subclause 5.9 3DMC extension tools:

5.9 3DMC extension tools

3DMC extension tools are basically based on 3D mesh coding (3DMC) tools introduced in MPEG-4 Visual [ISO/IEC 14496-2]. Compared to 3DMC tools, 3DMC extension tools incorporate vertex order and face order preserving functionality, efficient texture mapping functionality, and new stitching operation and remove forest split operation, computational graceful degradation (CGD), and existing stitching operation.

5.9.1 Introduction

3D Mesh Object

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

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

Single Resolution Mode

The incremental representation of a single resolution 3D model is based on the Topological Surgery scheme. For manifold triangular 3D meshes, the Topological Surgery representation decomposes the connectivity of each *connected component* into a *simple polygon* and a *vertex graph*. All the triangular faces of the 3D mesh are connected in the simple polygon forming a *triangle tree*, which is a spanning tree in the dual graph of the 3D mesh. Figure AMD1-4 shows an example of a triangular 3D mesh, its dual graph, and a triangle tree. The vertex graph identifies which pairs of boundary edges of the simple polygon are associated with each other to reconstruct the connectivity of the 3D mesh. The triangle tree does not fully describe the triangulation of the simple polygon. The missing information is recorded as a *marching edge*.

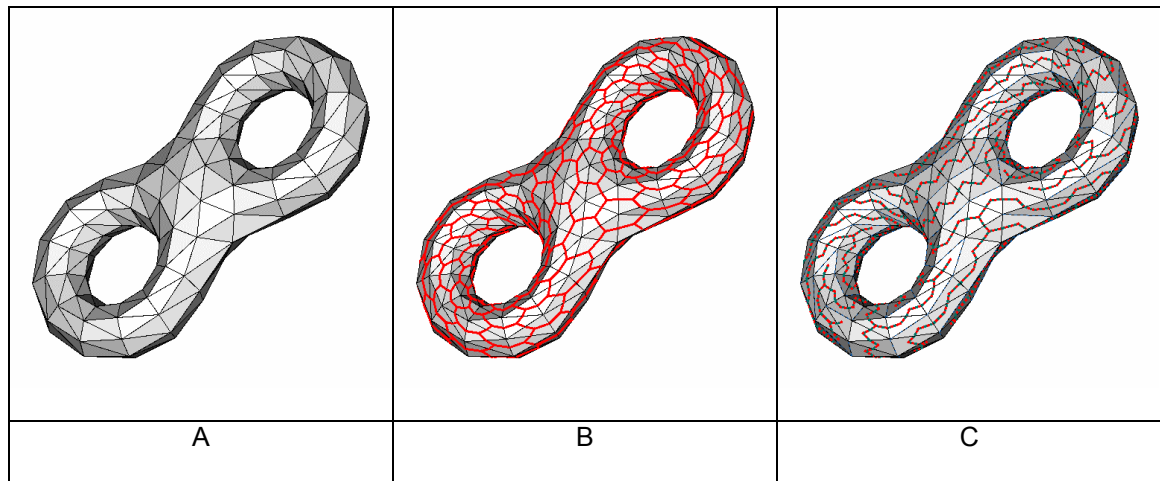


Figure AMD1-4 — A triangular 3D mesh (A), its dual graph (B), and a triangle tree (C)

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

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

Incremental Representation

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

Error Resilience for 3D Mesh Object

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

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

Vertex Order and Face Order Preserving

To animate or edit the content represented by IFS, one can do the operation per vertex. The authoring tool is assumed to use a fixed vertex order in a scene for easy and efficient handling of animation and updating. When 3DMC is used on IndexedFaceSet, this presumed order is broken for the IndexedFaceSet node. The encoder of 3DMC changes the vertex order to maximize the compression efficiency. This causes an additional problem when used with other tools that share a fixed vertex order. Not only 3DMC changes vertex order after compression, it also changes the face order. This may not be a problem if editing or animation of a 3D model is done per vertex, where vertex order is a problem to fix in such a case. However, if editing or animation is done per face, the change of face order may have the same impact as vertex order.

These vertex and face order changes may create a lot of confusion not only at the encoder side, but more at the decoder side. Hence, in order to solve this issue it needs to carry original vertex and face order information with encoded bitstream and re-order the vertex and face order accordingly after decoding the encoded model.

Efficient Texture Mapping

Efficient texture mapping would alleviate the need of having very accurate geometry model, since approximation by texture map will do the trick for the user. Therefore, the accuracy of texture coordinates is critical in order to guarantee the quality of rendered quality of 3D models.

Near lossless or lossless compression of texture coordinate, hence, is a very important issue to make sure. The current IndexedFaceSet-based representation describes the texture coordinates in float, where the texture coordinates in reality are discrete values in integer. To compress the texture coordinates losslessly from the point of integer values, two kinds of schemes can be used: (1) if the texture image size is previously known, quantised step size for texture coordinates can be set as the inverse of the texture image size; (2) if the size of texture image is not known, the possible quantised step size can be estimated by analyzing the difference values of the real texture coordinate values (or the regular intervals of ordered texture coordinate values).

Stitching for Non-Manifold and Non-Orientable Meshes

The connectivity of a non-manifold and non-orientable 3D mesh is represented as a manifold 3D mesh and a sequence of *stitches*. Each stitch describes the number of duplications for the vertex increase or face increase and the actual index of the original vertex or face and duplicated vertex (vertices) or face (faces) during the conversion of non-manifold and non-orientable into an oriented-manifold 3D mesh and a sequence of stitches.

Encoder and Decoder Block Diagrams

High level block diagrams of a general 3D polygonal model encoder and decoder are shown in Figure AMD1-5. They consist of a 3D mesh connectivity (de)coder, geometry (de)coder, property (de)coder, vertex/face order (de)coder, and entropy (de)coding blocks. Connectivity, vertex position, and property information are extracted from 3D mesh model described in VRML or MPEG-4 BIFS format. The connectivity (de)coder is used for an efficient representation of the association between each face and its sustaining vertices. The geometry (de)coder is used for a lossy or lossless compression of vertex coordinates. The property (de)coder is used for a lossy or lossless compression of colour, normal, and texture coordinate data. The vertex/face order (de)coder is used for vertex order and face order preserving.

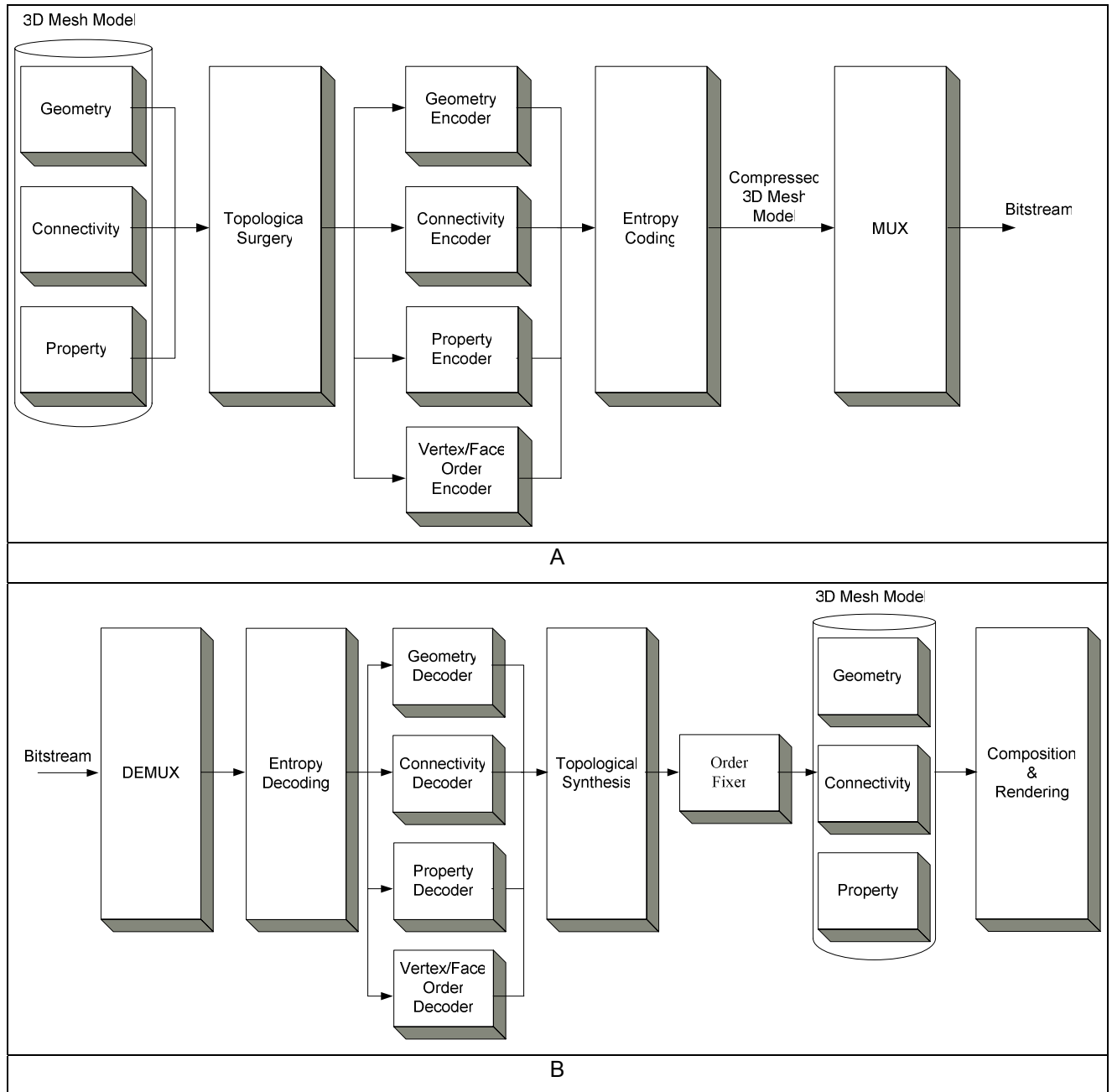


Figure AMD1-5 — General block diagram of the 3D mesh compression.

A: 3D mesh encoder.

B: 3D mesh decoder.

5.9.2 Visual bitstream syntax and semantics

5.9.2.1 Structure of coded visual data

5.9.2.1.1 3D Mesh Object

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

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

nCC is the number of connected components.

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

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

Additionally, if the vertex and face order preserving is supported, the last connected component data block is followed by one or two data blocks, each one of them representing vertex order and face order information. Vertex order and face order information can be applied to all the components of a bitstream.

Vertex Order	Face Order
--------------	------------

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

Vertex Graph	Triangle Tree	Triangle Data
--------------	---------------	---------------

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

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

The data for a given triangle is organized as follows:

marching edge	td_orientation	polygon_edge	coord	normal	color	texCoord
---------------	----------------	--------------	-------	--------	-------	----------