INTERNATIONAL STANDARD

ISO/IEC 14496-16

Fourth edition
2011-11-01

AMENDMENT 2
2014-01-15

Information technology — Coding of audio-visual objects —

Part 16: Animation Framework eXtension (AFX)

AMENDMENT 2: Multi-resolution 3D mesh compression

Technologies de l'information — Codage des objets audiovisuels —

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

AMENDEMENT 2: Compression de maillages 3D multirésolution

Reference number
ISO/IEC 14496-16:2011/Amd.2:2014(E)

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 2 to ISO/IEC 14496-16:2011 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 2: Multi-resolution 3D mesh compression

Add 5.2.6:

5.2.6 Multi-Resolution 3D Mesh Coding

Multi-Resolution 3D Mesh Coding (MR3DMC) specifies a progressive compression approach for manifold triangular 3D meshes providing efficient rate-distortion performances and supporting the following functionalities:

- Lossless connectivity coding: retrieve the original connectivity with a possible permutation of the mesh vertices/triangles.
- Spatial scalability: the mesh resolution (i.e. the number of triangles/vertices) is adapted to the terminal rendering performances and to the available bandwidth.
- Quality scalability: the precision of coordinates/attributes is progressively refined as the bitstream is decoded.
- Near-lossless encoding of the geometry/attributes: the maximal error permitted, when the entire bitstream is decoded, is controlled by varying a set of encoder parameters.

5.2.6.1 MR3DMC Bistream structure

The MR3DMC stream describes a multi-resolution representation of any triangular mesh stored as an IndexedFaceSet, with single or multiple attributes defined per vertex of per triangle. The stream is decoded in a coarse-to-fine way by exploiting a set of spatial and quality levels of detail (LODs). The bitstream is composed of two main components (*cf.* Figure AMD1.1):

- The header: describing general information about the coded mesh.
- The data stream: describing the mesh LODs.

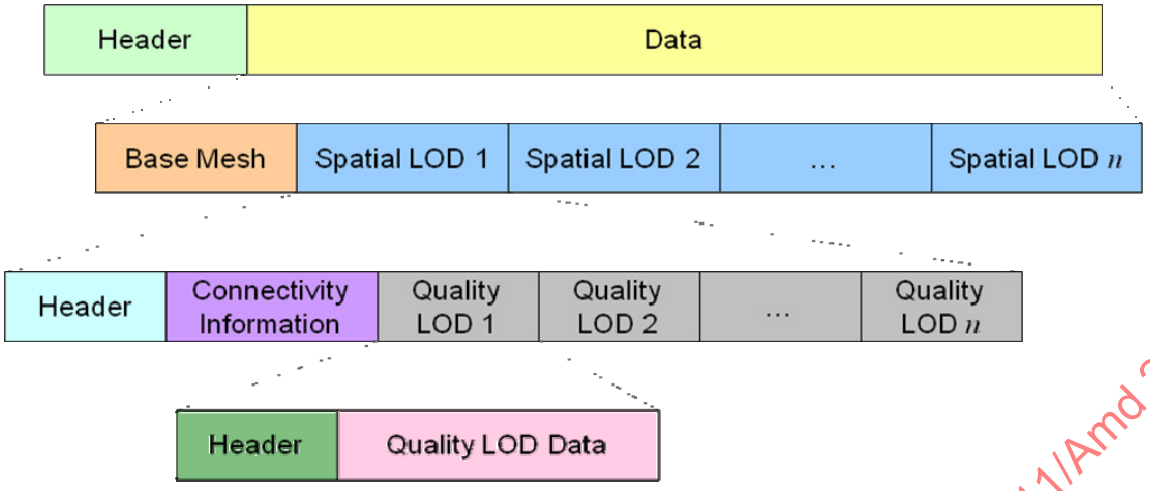

Figure AMD1.1 — MR3DMC stream structure

The MR3DMCStream is encapsulated in an AFX stream and has the following AFX object code:

Table AMD1.1 — AFX object code

AFX object code	Object	Associated node	Type value of bitwrapper
0x0C	Multi-Resolution 3D Mesh	IndexedFaceSet	2

When used in a BIFS scene, the value of the field "type" is 2.

5.2.6.2 MR3DMC Bitstream syntax and semantics

5.2.6.2.1 MR3DMC Bistream structure

5.2.6.2.1.1 Syntax

```
class MR3DMCStream {
    MR3DMCStreamHeader header;
    MR3DMCStreamData data;
}
```

5.2.6.2.1.2 Semantics

header: This is the header buffer of MR3DMC.

data: This is the data buffer of MR3DMC.

5.2.6.2.2 MR3DMCStreamHeader class

5.2.6.2.2.1 Syntax

```
class MR3DMCStreamHeader {

    unsigned int (32) streamSize;
```

```
bit (8) simplificationMode;
bit (16) numberOfSpatialLODs;
float (32) creaseAngle;
bit (1) ccw;
bit (1) solid;
bit (1) convex;
bit (1) colorPerVertex;
bit (1) normalPerVertex;
bit (1) otherAttributesPerVertex;
bit (1) isTriangularMesh;
bit (1) markerBit // always set as 1

unsigned int (32) numberOfCoord;
unsigned int (32) numberOfNormal
unsigned int (32) numberOfTexCoord;
unsigned int (32) numberOfColor;
unsigned int (32) numberOfOtherAttributes;
unsigned int (32) maxNumberOfVerticesInPatch


if (numberOfOtherAttributes >0) {
      unsigned int (8) dimensionOfOtherAttributes;
}
if (numberOfCoord>0) {
      unsigned int (32) numberOfCoordIndex;
      bit(8) QPforGeometry;
}
if (numberOfNormal>0) {
      bit(8) QPforNormal;
}
if (numberOfColor>0) {
      bit(8) QPforColor;
}
if (numberOfTexCoord>0) {
      bit(8) QPforTexCoord;
}
if (numberOfOtherAttributes >0) {
      bit(8) QPforOtherAttributes
}
if (numberOfCoord>0) {
      for(i=0;i<3;i++) {
            float(32) quantMinGeometry[i];
            float(32) quantRangeGeometry[i];
      }
}
if (numberOfNormal>0) {
      for (i=0;i<3;i++) {
            float(32) quantMinNormal[i];
            float(32) quantRangeNormal[i];
      }
}
if(numberOfColor>0) {
      for(i=0;i<3;i++) {
            float(32) quantMinColor[i];
            float(32) quantRangeColor[i];
      }
}
if(numberOfTexCoord>0) {
      for(i=0;i<2;i++) {
            float(32) quantMinTexCoord[i];
```

```
            float(32) quantRangeTexCoord[i];
        }
    }
    if(numberOfOtherAttributes>0) {
        for(i=0;i< dimensionOfOtherAttributes;i++) {
            float(32) quantMinOtherAttributes[i];
            float(32) quantRangeOtherAttributes[i];
        }
    }
    unsigned int (32) numberOfConnectedComponents;
};
```

5.2.6.2.2.2 Semantics

streamSize: A 32-bit unsigned integer describing the size in bytes of the current MR3DMC stream.

simplificationMode: A 8-bit unsigned integer indicating the simplification strategy

Table AMD1.2 — MR3DMC simplification modes

simplificationMode	Method
0	Geometry aware simplification
1	Connectivity-based simplification
2-255	ISO reserved

numberOfSpatialLODs: A 16-bit unsigned integer indicating the number of spatial LODs

creaseAngle: A 32-bit float indicating the IFS *creaseAngle* parameter which controls the default normal generation process.

ccw: 1-bit flag describing the IFS *ccw* parameter, which indicates whether the vertices are ordered in a counter-clockwise direction when the mesh is viewed from the outside.

solid: 1-bit flag describing the IFS *solid* parameter which indicates whether the shape encloses a volume.

convex: 1-bit flag describing the IFS *solid* parameter which indicates whether all faces in the shape are convex (should be always 1 for triangular meshes).

colorPerVertex: 1-bit flag describing the IFS *colorPerVertex* parameter which indicates whether the colors are defined per vertex.

normalPerVertex: 1-bit flag describing the IFS *normalPerVertex* parameter which indicates whether the normals are defined per vertex.

otherAttributesPerVertex: 1-bit flag describing whether the other attributes are defined per vertex.

isTriangularMesh: 1-bit flag describing whether the mesh is triangular (should be always 1).

markerBit: Always set as 1

numberOfCoord: A 32-bit unsigned integer indicating the number of position coordinates in the fine resolution mesh.**numberOfNormal**: A 32-bit unsigned integer indicating the number of normal coordinates in the fine resolution mesh.

numberOfTexCoord: A 32-bit unsigned integer indicating the number of texture coordinates in the fine resolution mesh.

numberOfColor: A 32-bit unsigned integer indicating the number of color coordinates in the fine resolution mesh.

numberOfOtherAttributes: A 32-bit unsigned integer indicating the number of the other attributes in the fine resolution mesh.

maxNumberOfVertiexInPatch: A 32-bit unsigned integer indicating the maximum number of vertices in the patches, which the spatial LOD consists of.

dimensionOfOtherAttributes: A 32-bit unsigned integer indicating the dimension (i.e., number of attributes) of the other attributes.

numberOfCoordIndex: A 32-bit unsigned integer indicating the number of faces associated to the position coordinates.

QPforGeometry: A 8-bit data indicating quantization parameter for geometry.

QPforNormal: A 8-bit data indicating quantization parameter for normals.

QPforColor: A 8-bit data indicating quantization parameter for colour.

QPforTexCoord: A 8-bit data indicating quantization parameter for texture coordinate.

QPforOtherAttributes: A 8-bit data indicating quantization parameter for other attributes

quantMinGeometry[]:1 by 3 array containing 32 bit floating data indicating minimum value used for geometry quantization

quantRangeGeometry[]: 1 by 3 array containing 32-bit floating point data indicating range values used for geometry quantization

quantMinNormal[]: 1 by 3 array containing 32 bit floating data indicating minimum value used for normal quantization

quantRangeNormal[]: 1 by 3 array containing 32-bit floating point data indicating range values used for normal quantization

quantMinColor[]: 1 by 3 array containing 32 bit floating data indicating minimum value used for color quantization

quantRangeColor[]: 1 by 3 array containing 32-bit floating point data indicating range values used for color quantization

quantMinTexCoord[]: 1 by 2 array containing 32 bit floating data indicating minimum value used for texcoord quantization

quantRangeTexCoord[]: 1 by 2 array containing 32-bit floating point data indicating range values used for texcoord quantization

quantRangeOtherAttributes[]: 1 by dimensionOfOtherAttributes array containing 32-bit floating point indicating range values used for normal quantization

numberOfConnectedComponents: A 32-bit unsigned integer indicating the number of the connected components.

5.2.6.2.3 MR3DMCStreamData class

5.2.6.2.3.1 Syntax

```
class MR3DMCStreamData {
    unsigned int (32) numberOfBaseVertices;
    unsigned int (32) numberOfBaseTriangles;
    TFANIndexDecoder(3, numberOfBaseVertices, numberOfBaseTriangles, 0, 0) baseMesh;
    for (int layer = 1; layer < numberOfSpatialLODs; layer++) {
        SpatialLODDecoder spatialLODdecoder;
    }
}
```

5.2.6.2.3.2 Semantics

numberOfBaseVertices: A 32-bit unsigned integer indicating the number of vertices in the base mesh.

numberOfBaseTriangles: A 32-bit unsigned integer indicating the number of triangles in the base mesh.

baseMesh: A TFAN strearm (cf. 5.2.5.3.9) describing the base mesh.

spatialLODdecoder: A stream specifying a spatial LOD layer.

5.2.6.2.4 SpatialLODDecoder Class

5.2.6.2.4.1 Syntax

```
class SpatialLODDecoder {
    SpatialLODDecoderHeader spatialLODDecoderHeader;
    ConnectivityLODDecoder connectivityLODDecoder;
    for (int qlayer = 0; qlayer < numberOfQualityLODs; qlayer++) {
        QualityLODDecoder qualityLODDecoder;
    }
}
```

5.2.6.2.4.2 Semantics

spatialLODDecoderHeader: Header buffer specifying the current spatial LOD properties.

connectivityLODDecoder: A ConnectivityLODDecoder stream describing the connectivity information of the current LOD.

qualityLODDecoder: A QualityLODDecoder stream describing the geometry refinement information for the different LODs associated with the current spatial LOD.

5.2.6.2.5 SpatialLODHeaderDecoder Class

5.2.6.2.5.1 Syntax

```
class SpatialLODHeaderDecoder
{
    bit (32) numberOfVertices;
```



```
    bit (32) numberOfTriangles;
    bit (8) numberOfQualityLayers;
    bit (8) qualityLODEncodingStrategy
}
```

5.2.6.2.5.2 Semantics

numberOfVertices: A 32 bit integer specifying the number of vertices in the current spatial LOD.

numberOfTriangles: A 32 bit integer specifying the number of triangles in the current spatial LOD.

numberOfQualityLODs: A 32 bit integer specifying the number of quality LODs in the current spatial LOD.

qualityLODEncodingStrategy: A 8-bit unsigned integer indicating the quality LOD encoding strategy

5.2.6.2.6 ConnectivityLODDecoder Class

5.2.6.2.6.1 Syntax

```
class ConnectivityLODDecoder
{
    if ( simplificationMode == 0) {
        TFANIndexDecoder(3, numberOfVertices, numberOfTriangles, 0, 0) connectivityCurrentLOD;
        IntArrayDecoder(numberOfVertices,1)    decodedVertexMapping;
    }
    else {
        for ( cc=0; cc < numberOfConnectedComponents; ++cc ) {
            unsigned int (32) numberOfRefinedVertices[cc]    = 0;
        }
        for ( cc = 0; cc < numberOfConnnectedComponents; ++cc ) {
            unsigned int(32) numberOfConnectivitySymbols;
            bit(8) conquestMode
            for ( int v = 0; v < numberOfConnectivitySymbols; ++v) {
                valenceOfRefinedVertices[v] = arithmetic_decoder.decode();
                if ( valenceOfRefinedVertices[v] > 0 ) {
                    numberOfRefinedVertices[cc]++;
                }
            }
        }
    }
}
```

5.2.6.2.6.2 Semantics

connectivityCurrentLOD: A TFANIndexDecoder stream (cf. 5.2.5.3.9s) describing the connectivity of the current spatial LOD.

decodedVertexMapping: A bitstream of type IntArrayDecoder specifying for each vertex of the previous LOD its index the current one.

numberOfConnectivitySymbols: An array of 32 bit unsigned integer specifying the number of connectivity symbols, which represent the valences of refined vertices or null in the current spatial LOD for each connected component.

conquestMode: A 8-bit unsigned integer indicating conquest strategy in the current spatial LOD.

Table AMD1.3 — MR3DMC conquest modes

conquestMode	Method
0	Decimating Conquest
1	Cleaning Conquest
2-255	ISO reserved

valenceOfRefinedVertices: An array of integer describing the valences of inserted vertices or null symbols in order to refine the previous spatial LOD.

numberOfRefinedVertices: An array of 32-bit unsigned integer indicating the number of the refined vertices for each spatial LOD

5.2.6.2.7 QualityLODDecoder class

5.2.6.2.7.1 Syntax

```
class QualityLODDecoder {
    QualityLODHeader qualityLODMR3DMCHeader
    QualityLODData qualityLODMR3DMCData

}
```

5.2.6.2.7.2 Semantics

qualityLODHeader: Header buffer specifying the current quality LOD properties.

qualityLODData: A QualityLODData stream describing the compressed predicted approximation errors associated with the current quality LOD vertices (cf. Annex U.7)

5.2.6.2.8 QualityLODHeader class

5.2.6.2.8.1 Syntax

```
class QualityLODHeader {
    if (qualityLODEncodingStrategy == 0 ) {
        bits(8) numberOfCurrentQPLayer;
        bits(32) numberOfVerticesOfCurrentQualityLOD;
    }
}
```

5.2.6.2.8.2 Semantics

numberOfCurrentQPLayer: An 8 bit integer indicating the number of QP layer in the current quality LOD.

numberOfVerticesOfCurrentQualityLOD: An 32 bit integer indicating the number of vertices in current quality LOD.

5.2.6.2.9 QualityLODData class

5.2.6.2.9.1 Syntax

```
class QualityLODData {
    If ( qualityLODEncodingStrategy == 0 ) {
        for ( j=0; j < qpLayerNumberOfCurrentQualityLOD; ++j ) {
            QPLayerLODDecoder qpLayerLODDecoder;
        }
    }
    else {
        for ( cc = 0; cc < numberOfConnectedComponents; ++cc ) {
            BitPlaneDecoder(numberOfRefinedVertices)  bitPlaneDecoder;
        }
    }
}
```

5.2.6.2.9.2 Semantics

qpLayerLODDecoder: A QPLayerLODDecoder stream describing the QP layer information of the current spaital LOD.

numberOfRefinedVertices: A 32-bit unsigned integer indicating the number of the refined vertices in the current spatial LOD.

itPlaneDecoder: A BitPlaneDecoder stream describing the bit plane information of the current spatial LOD.

5.2.6.2.10 QPLayerLODDecoder class

5.2.6.2.10.1 Syntax

```
class QPLayerLODDecoder {
    QPLayerLODHeader qpLayerLODHeader
    QPLayerLODData qpLayerLODData
}
```

5.2.6.2.10.2 Semantics

qpLayerLODHeader: Header buffer specifying the current QP Layer LOD properties.

qpLayerLODData: A QPLayerLODData stream describing the cascaded quantization based compressed predicted approximation errors associated with the current QP Layer vertices.

5.2.6.2.11 BitPlaneDecoder class

5.2.6.2.11.1 Syntax

```
class BitPlaneDecoder {
    bit (QPforGeometry) quantizedValue[numberOfRefinedVertices];
    bit (numberOfRefinedVertices) signValue;
    bit (8) transformMode;

    Adaptive_Data_Model mSign(2);
    Adaptive_Data_Model mCluster1[QPforGeometry ];
```

```
    Adaptive_Data_Modle mCluster2(2);
    Arithmetic_Codec acd(code_bytes, code_buffer);
    acd.start_decoder();
    for ( i = 0; i < QPforGeometry; ++I ) {
        mCluster1[i]. set_alphabet(2);
    }
    for ( i = 0; i < numberOfRefinedVertices; ++i ) {
        ACDecoder(mSign)   signValue;
    }

    AttributeDecoder(numberOfRefinedVertices)       attributDecoder;

    for ( i = 0; i < QPforGeometry; ++i ) {
        for (j = 0; j < numberOfRefinedVertices*3; ++j ) {
            if (quantizedValue[j] == 0 ) {
                quantizedValue[j][i] = aithmetic_decoder.decode(mCluster1[i]);
            }
            else {
                quantizedValue[j][i] = aithmetic_decoder.decode(mCluster2);
            }
        }
    }
}
```

5.2.6.2.11.2 Semantics

attributeDecoder: An AttributeDecoder stream describing single or multiple attributes defined per refined vertex of per refined triangle such as normals, colors, texture coordinates in the current quality LOD.

quantizedValue: A array of QPforGeometry-bit integers of dimension numberOfRefinedVertices indicating for the quantized values in the current spatial LOD.

signValue: Sign information for the quantized values in the current spatial LOD.

transformMode: A 8-bit unsigned integer indicating the transformation strategy

Table AMD1.4 — MR3DMC transformation modes

Value	Transformation Strategy
if (transformMode & 0x001 == 1)	KL transformation for x coordinate
if (transformMode & 0x010 == 2)	KL transformation for y coordinate
if (transformMode & 0x100 == 4)	KL transformation for z coordinate

5.2.6.2.12 QPLayerLODHeader class

5.2.6.2.12.1 Syntax

```
class QPLayerLODHeader {
    bits(8) qpValueOfCurrentQPLayerLOD ;
}
```

5.2.6.2.12.2 Semantics

qpValueOfCurrentQPLayerLOD : An 8 bit integer indicating the QP value for a given QP layer LOD.

5.2.6.2.13 QPLayerLODData class

5.2.6.2.13.1 Syntax

```
class QPLayerLODData {
    IntArrayDecoder(numberOfVerticesOfCurrentQualityLOD*3, 1) decodedQuantizedVerticesResidue;
}
```

5.2.6.2.13.2 Semantics

decodedQuantizedVerticesResidue: An array of decoded quantized vertices residue of the QP layer in the current Quality layer LOD.

5.2.6.2.14 AttributeDecoder class

5.2.6.2.14.1 Syntax

```
class AttributeDecoder
{
   if ( numberOfNormal != 0 )
   {
      if ( normalPerVertex == 1 )
      {
         unsigned int (32)    numberOfRefinedNormal;
         DecodefloatArray(numberOfRefinedNormal, 3, quantMinNormal, quantRangeNormal, tQP)
            decodedNormal;
      }
      else
      {
         DecodefloatArray(numberOfRefinedVertices, 3, quantMinNormal, quantRangeNormal, tQP)
         decodedNormal;
      }
   }

   if ( numberOfColor != 0 )
   {
      if ( colorPerVertex == 1 )
      {
         unsigned int (32)    numberOfRefinedColor;
         DecodedfloatArray(numberOfRefinedColor, 3, quantMinColor, quantRangeColor, tQP)   decodedColor
      }
      else
      {
         DecodefloatArray(numberOfRefinedColor, 3, quantMinColor, quantRangeColor, tQP) decodedColor
      }
   }
```

```
  if ( numberOfTexCoord != 0 )
  {
    unsigned int (32)  numberOfRefinedTexCoord;
    Decodefloat(numberOfRefinedTexCoord,  2,  quantMinColor,  quantRangeColor,  tQP)
        decodedTexCoord;
  }

  if ( numberOfOtherAttributes != 0 )
  {
    if ( otherAttributesPerVertex == 1 )
    {
      unsigned int (32)    numberOfRefinedOtherAttributes;
      DecodefloatArray(numberOfRefinedOtherAttributes,                dimensionOfOtherAttributes,
      quantMinOtherAttributes, quantRangeOtherAttributes, tQP)   decodedOtherAttributes
    }
    else
    {
      DecodefloatArray(numberOfRefinedVertices,  dimensionOfOtherAttributes,  quantMinOtherAttributes,
      quantRangeOtherAttributes, tQP) decodedOtherAttributes
    }
  }
}
```

5.2.6.2.14.2 Semantics

numberOfRefinedNormal: A 32-bit unsigned integer describing the number of refined normals in the current quality LOD.

decodedNormal: A reconstructed normal whose size is 1 by numberOfRefinedNormal*3.

numberOfRefinedColor: A 32-bit unsigned integer describing the number of refined colors in the current quality LOD.

decodedColor: A reconstructed normal whose size is 1 by numberOfRefinedColor*3

numberOfRefinedTexCoord: A 32-bit unsigned integer describing the number of refined texture coordinates in the current quality LOD.

decodedTexCoord: A reconstructed normal whose size is 1 by numberOfRefinedTexCoord*2

numRefinedOtherAttributes: A 32-bit unsigned integer describing the number of refined other attributes in the current quality LOD.

decodedOtherAttributes: A reconstructed other attributes whose size is 1 by numberOfOtherAttributes* dimensionOfOtherAttributes.

5.2.6.3 Decoding process

5.2.6.3.1 Overview

This clause specifies the decoding process that the decoder shall perform to recover 3D mesh data from the encoded bitstream. As shown in Figure AMD1.2, the decoding process includes a switch which indicates the algorithm used in the encoding process. In this document, only triangular meshes are considered.

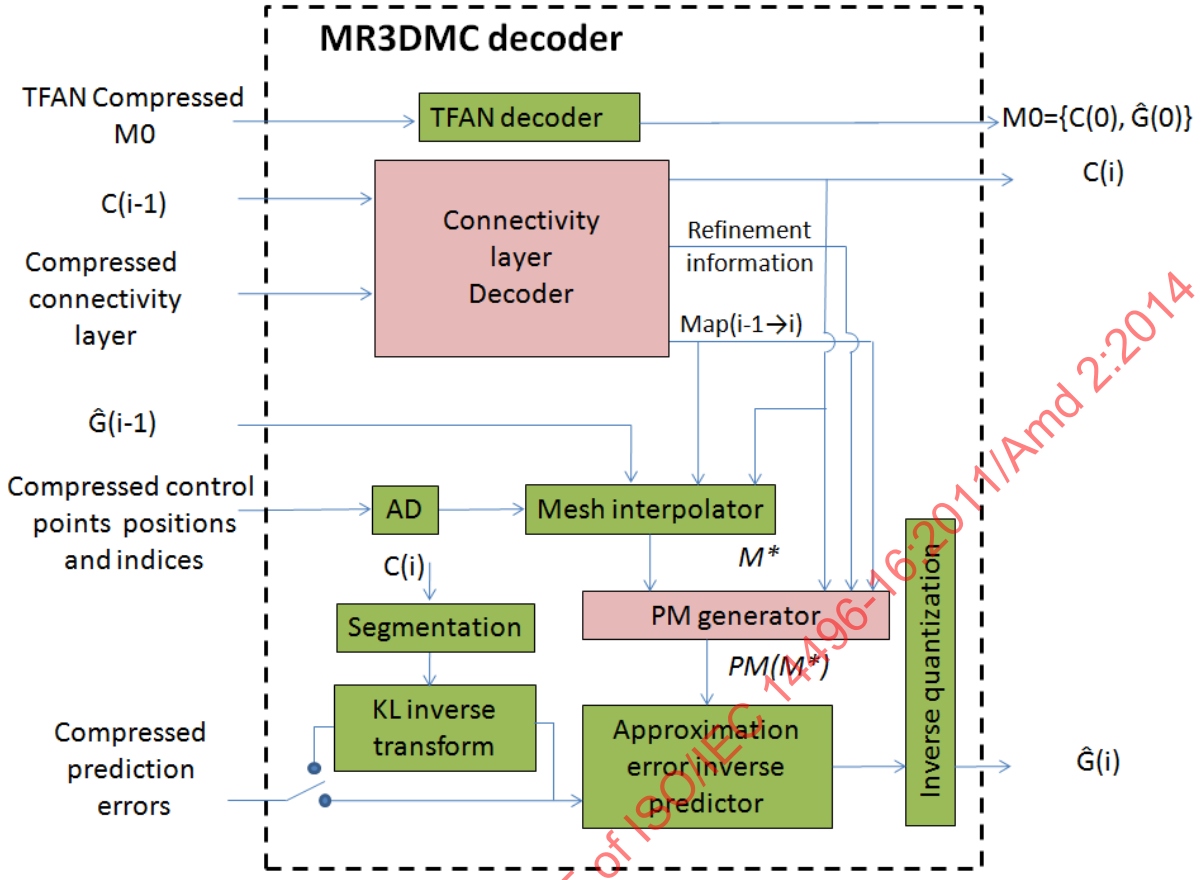

Figure AMD1.2 — MR3DMC decoding process

Figure AMD1.2 illustrates the MR3DMC decoding process.

5.2.6.3.2 Connectivity decoding

The base connectivity layer is decoded by exploiting the TFAN decoding procedure (cf. 5.2.5.4.4). The connectivity enhancement layers are decoded by using either the TFAN connectivity decoder or the valence-based connectivity decoder [1]. Here, the choice of the decoding strategy depends on the decimation strategy. If the geometry aware simplification mode is selected then the TFAN decoder is applied. Otherwise, if the connectivity-based simplification mode is considered then the valence-based decoding technique is exploited. The TFAN connectivity encoding and decoding procedures are described in 5.2.5.4.4 and Annex P, respectively.

The valence-based decoding process starts with an initial seed gate (an oriented edge) and refines the patch, which is adjacent to the gate, using the decoded degree of decimated vertex. Then the boundary edges of the refined patch are push onto a first-in-first-out queue. Until all decimated vertices are refined, we repeat the process with queue.

A detailed description of the valence-based encoder is provided in Annex U.1.

5.2.6.3.3 Inverse binarization

4.2.5.2.18.2.2.1 Cascaded quantization

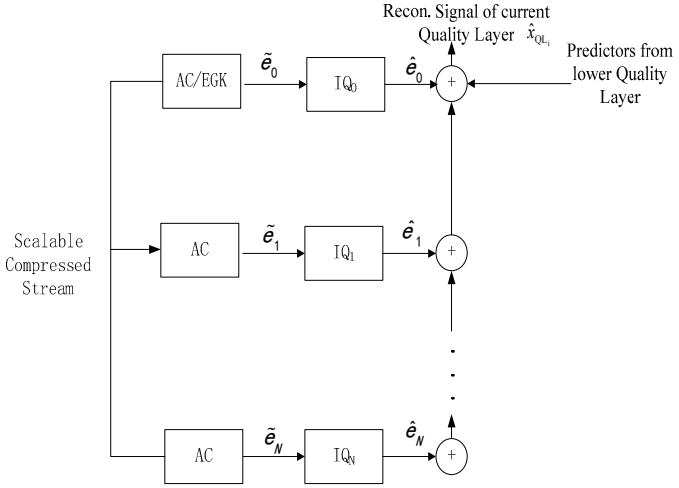

Figure AMD1.3 — An illustration of the cascaded inverse quantization

Figure AMD1.3 illustrates the decoding processing when only partial stream for a spatial LOD is received. The decoding processing is described as:

The predicted error for a quality layer can be calculated by

$$\hat{e} = \sum_{n=0}^{n=N'} \hat{e}_n ,$$

where N' is the highest quantization layer for current quality LOD for the received entire/partial stream. And $\hat{e}_n$ can be calculated by

$$|\hat{e}'_n| = |\hat{e}_n| \cdot VF << qbits$$
$$|\hat{e}_n| = (|\hat{e}'_n| + 32) >> 6$$
$$sign(\hat{e}_n) = sign(\tilde{e}_n)$$

where $<<$ indicates a binary left shift and VF=(Qstep $<<$ 6). The values of VF for $0 \le QP \le 5$ are defined in the standard as follows:

QP	0	1	2	3	4	5
VF	40	44	52	56	64	72

Since VF is a period of 6, other VFs relative to QPs can be derived accordingly.

5.2.6.3.4 Bit plane decoding

In order to obtain the uniformly quantized vectors, which consist of the tangential components or the transform coefficients for normal components, the MR3DMC decodes the bit plane, which was encoded with the context-based arithmetic codec, in the raster scan order from the most significant bit (MSB) planes to the least significant bit (LSB) planes. The bits are classified into two clusters. In each column, the bits from the MSB to the first '1' bit belong to the 1st cluster, while the other bits compose the 2nd cluster.

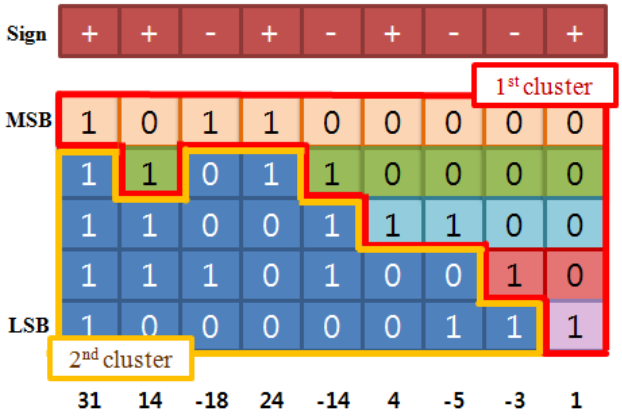

Figure AMD1.4 – Bit plane coding

5.2.6.3.5 Inverse KL transform

For each quantized vector, the MR3DMC decodes the bits from the 1st cluster until the quantized vector is zero and then decodes the bits from the 2nd cluster.

In order to apply the inverse KL transform to the coefficients, the MR3DMC decoder computes the topological distance between every pair of decimated vertices and defines the covariance matrix $\mathbf{C}$ by

$$\mathbf{C} = \begin{bmatrix} 1 & \rho^{d_{1,2}} & \cdots & \rho^{d_{1,n}} \\ \rho^{d_{2,1}} & 1 & & \rho^{d_{2,n}} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^{d_{n,1}} & \rho^{d_{n2}} & \cdots & 1 \end{bmatrix},$$

where $d_{i,j}$ is the topological distance between decimated vertex i and j. Since $\mathbf{C}$ is real and symmetric, its eigen-value decomposition is possible, which is given by

$$\mathbf{C} = \mathbf{U}\Lambda\mathbf{U}^T,$$

where Λ is the diagonal matrix, whose diagonal elements are the eigen-values λ_i (i = 1, 2, . . . , n) arranged in the decreasing order of the magnitudes. Also, the matrix $\mathbf{U}$ has column vectors, which are the eigenvectors. The inverse KL transform is given by the matrix $\mathbf{U}$ and the decoder can construct the normal components from the coefficient vector through the inverse KL transform by

$$x = \mathbf{U}y,$$

where x is the normal component and y is the coefficient of KL transform.

5.2.6.3.6 Inverse Prediction

The MR3DMC decoder progressively decompresses the approximation prediction errors starting from the lower LOD to the higher one. Here, at each step, the three components $\hat{e}_v^n$, $\hat{e}_v^t$ and $\hat{e}_v^r$ of the predicted approximation errors are arithmetically decoded, de-quantized and then used to reconstruct the approximation error $\hat{e}_v$, as follows:

$$\hat{e}_v = \hat{e}_v^n \cdot \hat{n}_v^* + \hat{e}_v^t \cdot \hat{t}_v^* + \hat{e}_v^r \cdot \hat{r}_v^*,$$

where n_v^* is the normal of M^* at vertex v, and r and t are two vectors chosen to form a direct orthonormal basis with n_v^*.

Finally, the decoded positions $\left(\hat{P}_v\right)$ are given by:

$$\hat{P}_v = \hat{e}_v + \frac{1}{v^*}\sum\nolimits_{w \in v^*} \hat{P}_w + \left(P_v^* - \frac{1}{v^*}\sum\nolimits_{w \in v^*} P_w^* \right) \quad (1)$$

Note that by encoding/decoding the vertices in the reverse order of *PM(M*)*, the MR3DMC encoder/decoder guaranty that when processing the vertex v, the positions $\left(\hat{P}_w\right)_{w \in v^*}$ of all its neighbors have already been reconstructed. Furthermore, by exploiting the progressive mesh structure *PM(M*)*, the MR3DMC codec directly supports the spatial scalability functionality. The quality scalability is obtained by reconstructing all the vertex positions, while setting to zero the non-decoded predicted approximation errors $\hat{e}_v$ in equation (1).

5.2.6.3.7 Segmentation

Before applying the KL transform, the MR3DMC decoder partitions the current spatial LOD into a number of patches only using the connectivity information.

In the base mesh, each triangle is corresponded into each patch. During the refinement, the numbers of vertices in each patch increase. If a patch contains more than the pre-defined number, **maxNumberOfVerticesInPatch** of vertices, then the MR3DMC decoder divides the patch into two patches.

5.2.6.3.8 Mesh approximation

The Laplacian matrix *L* is defined as follows:

$$\forall (i,j) \in \{1,\cdots,V+C\} \times \{1,\cdots,V\}\; L_{i,j} = \begin{cases} 1 & \text{if } j = i \\ -\dfrac{\alpha}{\alpha\left|i_S^*\right| + \beta\left|i_n^*\right|} & \text{if } i \le V \text{ and } j \in i_S^* \\ -\dfrac{\beta}{\alpha\left|i_S^*\right| + \beta\left|i_n^*\right|} & \text{if } i \le V \text{ and } j \in i_n^* \\ 1 & \text{if } j \in C \\ 0 & \text{otherwise} \end{cases}$$

where,

- i_S^* is the set of topological neighbors of the vertex i sharing with it either a boundary edge (i.e. adjacent to exactly one triangle) or a salient edge (i.e. it belongs to S),
- i_n^* is the set of the neighbors of the vertex *i* not belonging to i_S^*,
- $\left|i_S^*\right|$ and $\left|i_n^*\right|$ are the number of elements of i_S^* and i_n^* respectively,
- α and β are the weights associated to the special edges (i.e. salient or boundary edges) and to the non-special ones, respectively.

The $V \times 3$ matrix of approximated vertex positions, denoted $P^* = \left(P_v^*(k)\right)_{v \in \{1, \ldots, V\}}^{k \in \{1, 2, 3\}}$, is computed by solving the following sparse linear system:

$$\left(L^T L\right) P^* = B.$$

The $V \times 3$ matrix B is given by:

$$B_{i,k} = \begin{cases} P_i(k) & \text{if } \left(i \in C\right) \\ 0 & \text{otherwise} \end{cases}$$

where $P_i(1)$, $P_i(2)$ and $P_i(3)$ represent the original Cartesian x, y and z coordinates of the vertex i, respectively.

5.2.6.3.9 Multiple Attributes

During the decoding process, the base mesh M^0 is gradually refined by the sequence of vertex insertions and re-triangulations (see Figure U-7). For each vertex insertion, attributes of vertices or triangles, which are adjacent to the inserted vertex, should be updated or decoded. The MR3DMC decoder decompresses single or multiple attributes defined per updated vertex of per triangle (See Figure AMD1.5).

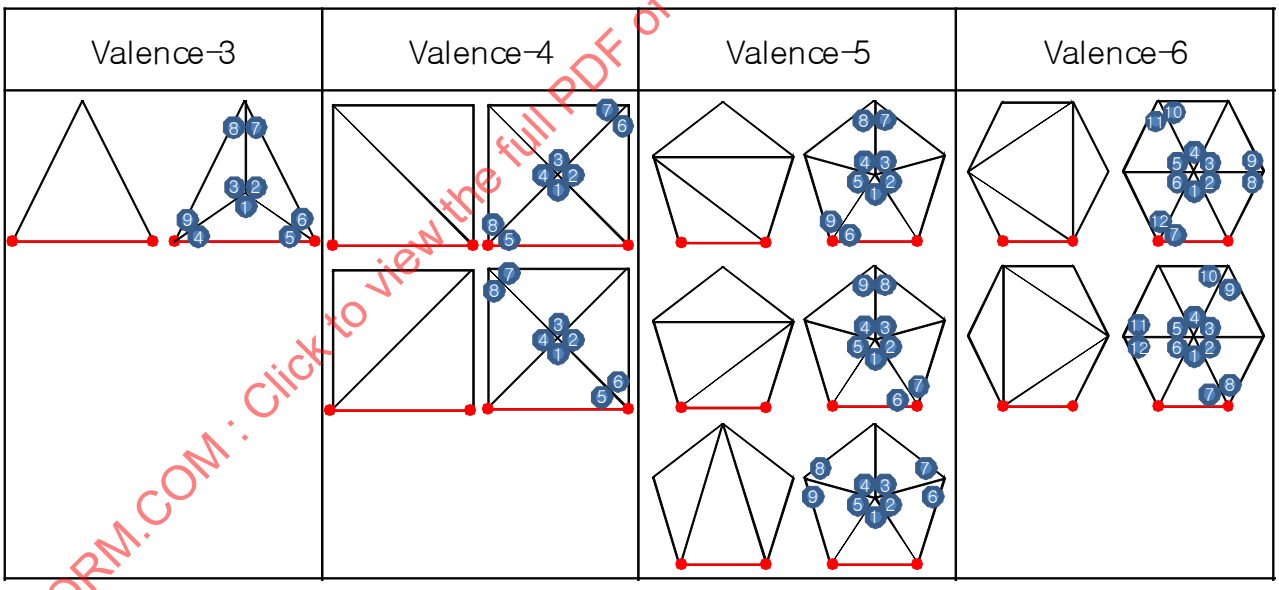

Figure AMD1.5 — Decoding order of attribute per vertex

Insert Annex U, and update the numbering of Annexes accordingly:

Annex U
(informative)

MR3DMC Encoding Process

U.1 General

Figure U.1 illustrates the MR3DMC encoding process.

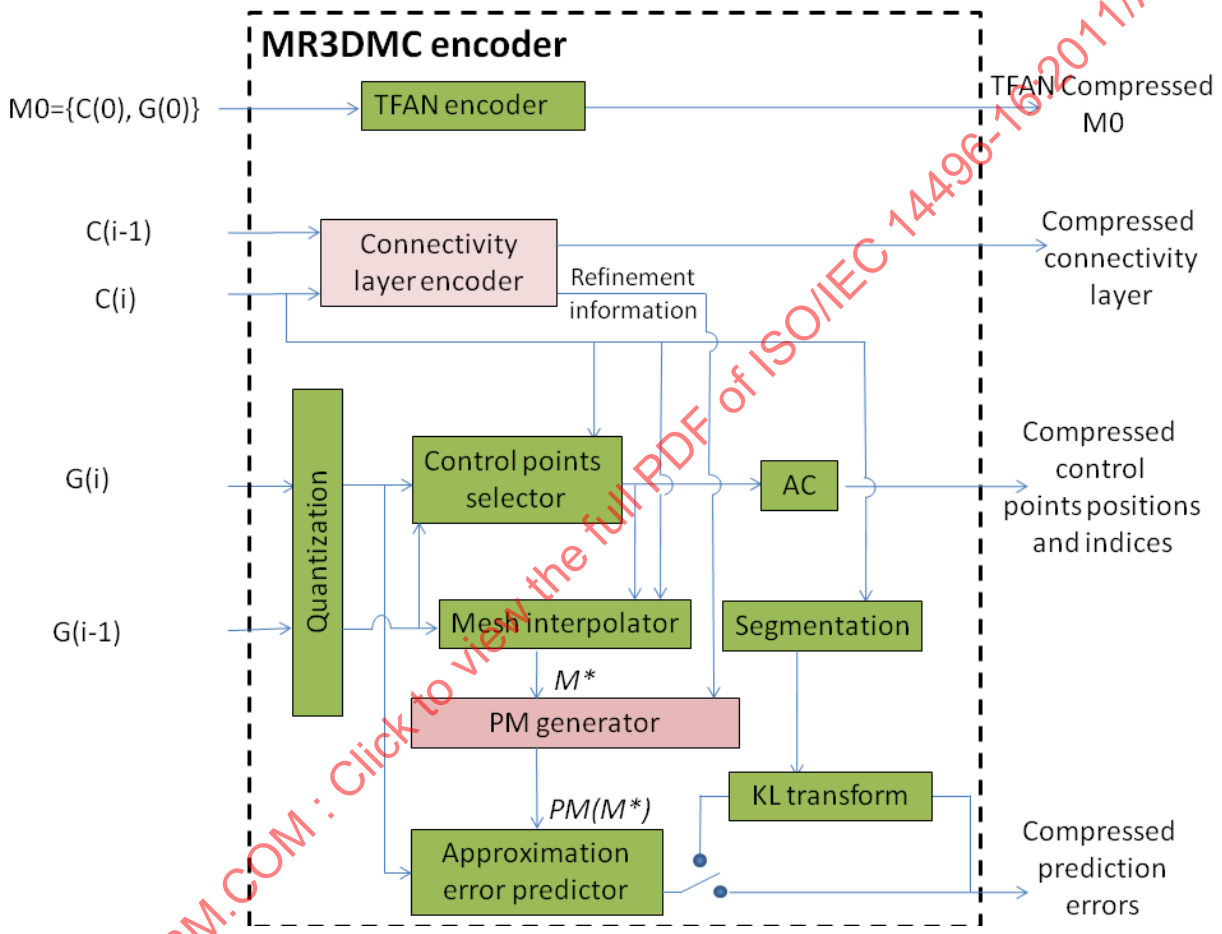

Figure U.1 — MR3DMC encoding process

The MR3DMC processes the input mesh in order to generate:

- a set of connectivity LODs, denoted (LOD(i)), and
- a set of LOD mapping information, denoted Map(i-1 $\rightarrow i$), defining a correspondence between the vertices of each couple of consecutive LODs (i.e. LOD(i-1) and LOD(i)).

The base mesh M0, which corresponds to LOD(0), is composed of connectivity C(0) and geometry G(0). M0 is encoded by directly exploiting the TFAN codec. The remaining LODs are progressively encoded by using either the TFAN connectivity encoder (cf. Annex P) or the valance-based techniques described in Annex P.1.

The connectivity information of LOD(i), denoted C(i), is used in order to progressively compress the geometry information. More precisely, the MR3DMC technique exploits the mesh connectivity in order to derive a smooth approximation of the current LOD, denoted M*. M* is entirely described by:

- the connectivity information C(i) of LOD(i),
- the geometry information of C(i-1) of LOD(i-1),
- the mapping information Map(i-1 → i) mapping the vertices of LOD(i-1) to those of LOD(i),
- a small set of N control points denoted $CP = (c_k)_{k\in\{1,\cdots,N\}}$, and
- the set of indices S of the salient edges.

The positions $CP = (c_k)_{k\in\{1,\cdots,N\}}$ of the control points are quantized and arithmetically encoded together with the indices *CP* and *S*. A progressive mesh hierarchy, denoted $PM(M^*)$, is then constructed by decimating M^*, allowing only half-edge collapse operations. $PM(M^*)$ is then exploited in order to predict the approximation errors of M^* w.r.t. M. Finally, the predicted approximation errors $(e_v)_{v\in\{1,\cdots,V\}}$ (V being the number of vertices) are arithmetically encoded and progressively transmitted to the decoder.

U.2 Valence-based connectivity encoding

To generate the multi-resolution 3D mesh, the input mesh M is successively simplified to a series of layers with until the base layer M^0 is obtained. Here a sequence of vertex removals followed by local re-triangulation is performed (see Figure U-7). The decimated vertices are not adjacent to each other, when the simplification from M to M^0 is performed.

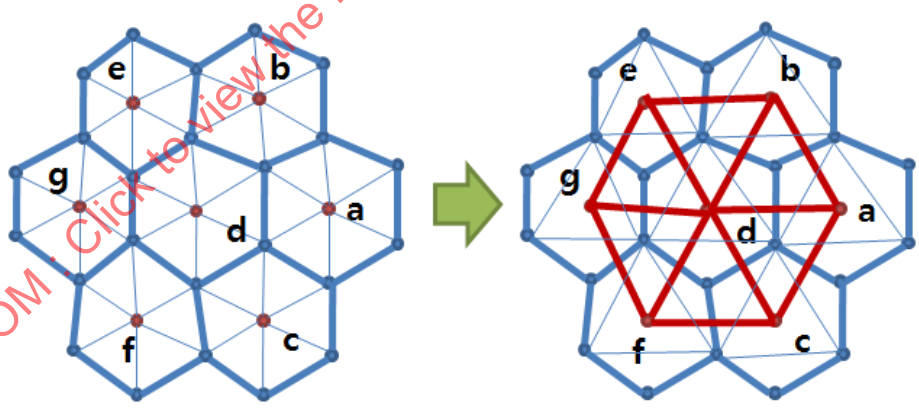

Figure U.2 — Mesh Simplification and Connectivity Information between Decimated Vertices

After simplification step, each layer consists of the valence sequence for the connectivity reconstruction and normal and tangential components of the decimated vertices except for the base layer.

U.3 Laplacian Mesh Approximation

The MR3DMC codec defines the Laplacian matrix *L* as follows:

$$\forall (i,j) \in \{1,\cdots,V+C\} \times \{1,\cdots,V\}\, L_{i,j} = \begin{cases} 1 & \text{if } j = i \\ -\dfrac{\alpha}{\alpha \left|i_S^*\right| + \beta \left|i_n^*\right|} & \text{if } i \le V \text{ and } j \in i_S^* \\ -\dfrac{\beta}{\alpha \left|i_S^*\right| + \beta \left|i_n^*\right|} & \text{if } i \le V \text{ and } j \in i_n^* \\ 1 & \text{if } j \in C \\ 0 & \text{otherwise} \end{cases}$$

where,

- i_S^* is the set of topological neighbors of the vertex i sharing with it either a boundary edge (i.e. adjacent to exactly one triangle) or a salient edge (i.e. it belongs to S),
- i_n^* is the set of the neighbors of the vertex *i* not belonging to i_S^*,
- $\left|i_S^*\right|$ and $\left|i_n^*\right|$ are the number of elements of i_S^* and i_n^* respectively,
- α and β are the weights associated to the special edges (i.e. salient or boundary edges) and to the non-special ones, respectively.

Different procedures could be considered to detect salient edges. A possible solution is to consider as salient all the edges exhibiting a dihedral angle higher than a user fixed threshold (e.g. $\frac{\pi}{6}$).

The $V \times 3$ matrix of approximated vertex positions, denoted $P^* = \left(P_v^*(k)\right)_{v \in \{1,\ldots,V\}}^{k \in \{1,2,3\}}$, is computed by solving the following sparse linear system:

$$\left(L^T L\right) P^* = B.$$

The $V \times 3$ matrix B is given by:

$$B_{i,k} = \begin{cases} P_i(k) & \text{if } (i \in C) \\ 0 & \text{otherwise} \end{cases}$$

where $P_i(1)$, $P_i(2)$ and $P_i(3)$ represent the original Cartesian x, y and z coordinates of the vertex i, respectively.

Note that if the weights α and β are equal or if there are no special edges, we obtain exactly the definition of the Laplacian matrix proposed in [Chen'05]. A possible choice is to set α to 100 and β to 1. This implies that a vertex *i* located on a mesh boundary or on a salient edge is a *100* times more influenced by its special neighbors i_S^* than by the non-special ones (i.e. i_n^*).

U.4 Progressive Mesh Hierarchy Generation

The *Progressive Mesh* (PM) technique represents the original mesh M as a base mesh M_0 together with a set of *vertex split* refinement operations (*cf.* Figure 3). The coarse mesh M_0 is obtained by successively decimating M with edge-collapse decimation operations (*cf.* Figure 3). The choice of the sequence of the edge-collapse operations to be applied to *M* is usually guided by a shape preservation strategy. Here, at each step of the decimation process, the cost of applying each possible edge collapse operation is evaluated in order to choose the one which induces the lowest shape distortions. By exploiting the fact that each edge-collapse operation is invertible, a set of refinement operations, denoted $(vsplit_i)_{i\in\{1,\ldots,L\}}$ (*L* being the number of Level of Details (LODs)), can then be computed and exploited to progressively regenerate M starting from M_0.

U.5 Approximation errors prediction

The MR3DMC encoder compresses the predicted approximation errors in the reverse order of *PM(M*)*. At each step, the predicted approximation error e_v associated with the vertex v is computed as follows:

$$e_v = \left(P_v - \frac{1}{v^*}\sum_{w\in v^*}\hat{P}_w\right) - \left(P_v^* - \frac{1}{v^*}\sum_{w\in v^*}P_w^*\right)$$

where v^* represents the set of topological neighbors of v in the current LOD of *PM(M*)* and $\left(\hat{P}_w\right)_{w\in v^*}$ are the positions reconstructed by the decoder as described in the following (*cf.* equation (1)).

The obtained error e_v is then decomposed into a normal component e_v^n and two tangential ones e_v^t and e_v^r defined by:

$$e_v^n = e_v \cdot n_v^* \;,\; e_v^t = e_v \cdot t_v^* \;,\; e_v^r = e_v \cdot r_v^* \;,$$

where n_v^* is the normal of M^* at vertex v and r and t are two vectors chosen to form a direct orthonormal basis with n_v^*.

Finally, e_v^n, e_v^t and e_v^r are quantized and arithmetically encoded. Since, the normal component e_v^n contains usually more information than the tangential ones (i.e. e_v^t and e_v^r), it is quantized more finely.

U.6 Segmentation

In the segmentation step, we partition layer l into a number of segments, so that each segment contains less than the pre-defined number of vertices. During the segmentation step, the connectivity information between vertices is used and segments, which contain more vertices than the pre-defined number, will be divided into two segments.

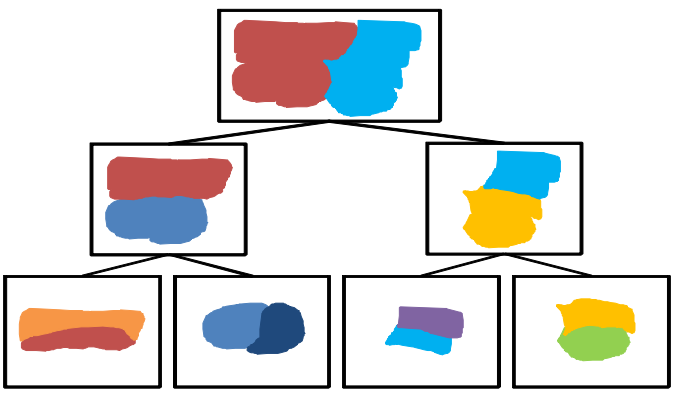

Figure U.3 — 3D mesh segmentation using connectivity between vertices