FINAL
DRAFT

AMENDMENT

ISO/IEC 14496-16:2009 FDAM 1

ISO/IEC JTC **1**

Secretariat: **ANSI**

Voting begins on:
2010-05-18

Voting terminates on:
2010-07-18

Information technology — Coding of audio-visual objects —

Part 16: Animation Framework eXtension (AFX)

AMENDMENT 1: Scalable complexity 3D mesh coding

Technologies de l'information — Codage des objets audiovisuels —

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

AMENDEMENT 1: Codage de maillage 3D de complexité échelonnable

Reference number
ISO/IEC 14496-16:2009/FDAM 1:2010(E)

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 14496-16:2009 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: Scalable complexity 3D mesh coding

Page 2, immediately before 3.1

Add the following:

4C	4-bits-based coding
AC	Arithmetic Coding
BPC	Bit Precision Coding
SVA	Shared Vertex Analysis
TFAN	Triangle FAN
QBCR	Quantization Based Compact Representation

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

Arithmetic operators

+	Addition.
-	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment. i.e. *x*++ is equivalent to *x* = *x* + 1
- -	Decrement. i.e. *x*-- is equivalent to *x* = *x* - 1
× *	Multiplication.
^	Power.
/	Integer division with truncation of the result toward zero. For example, 7/4 and -7/-4 are truncated to 1 and -7/4 and 7/-4 are truncated to -1.
÷	Used to denote division in mathematical equations where no truncation or rounding is intended.

% Modulus operator. Defined only for positive numbers.

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

Logical operators

|| Logical OR.

&& Logical AND.

! Logical NOT.

Relational operators

> Greater than.

>= Greater than or equal to.

≥ Greater than or equal to.

< Less than.

<= Less than or equal to.

≤ Less than or equal to.

== Equal to.

!= Not equal to.

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

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

Bitwise operators

& AND

| OR

>> Shift right with sign extension.

<< Shift left with zero fill.

Assignment

= Assignment operator.

Page 182, immediately before 4.3

Add the following new subclause.

4.2.5 Scalable Complexity 3D Mesh Coding

4.2.5.1 SC3DMC Bitstream structure

The Scalable Compression 3D Mesh Coding (SC3DMC) stream describes any triangular mesh represented as an IndexedFaceSet, with single or multiple attributes defined per vertex of per triangle. 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 connectivity and the geometry information of the mesh.

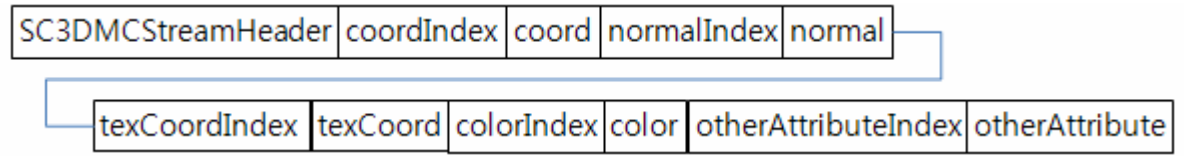

Figure AMD1.1 — SC3DMC stream structure.

The SC3DMCStream 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	Scalable complexity Based Representation	IndexedFaceSet	2

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

4.2.5.2 SC3DMC Bitstream syntax and semantics

4.2.5.2.1 SC3DMCStream class

4.2.5.2.1.1 Syntax

```
class SC3DMCStream{
    SC3DMCStreamHeader header;
    SC3DMCStreamData data;
}
```

4.2.5.2.1.2 Semantics

SC3DMCStreamHeader: This is the header buffer of SC3DMC.

SC3DMCSteamData: This is the data buffer of SC3DMC.

4.2.5.2.2 SC3DMCStreamHeader class

4.2.5.2.2.1 Syntax

```
class SC3DMCStreamHeader{

    unsigned int (32) streamSize;
    bit (8) encodingMode;
    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;


    if (numberOfOtherAttributes >0)
        unsigned int (8) dimensionOfOtherAttributes;
    if (numberOfCoord>0) {
        unsigned int (32) numberOfCoordIndex;
        bit(8) QPforGeometry;
    }
    if (numberOfNormal>0) {
        unsigned int (32) numberOfNormalIndex;
        bit(8) QPforNormal;
    }

    if (numberOfColor>0) {
        unsigned int (32) numberOfColorIndex;
        bit(8) QPforColor;
    }

    if (numberOfTexCoord>0) {
        unsigned int (32) numberOfTexCoordIndex;
        bit(8) QPforTexCoord;
        unsigned int (32) TexCoordWidth;
        unsigned int (32) TexCoordHeight;

    }
    if (numberOfOtherAttributes >0) {
        unsigned int (32) numberOfOtherAttributesIndex;
        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];
        }
    }
};
```


4.2.5.2.2.2 Semantics

streamSize: A 32-bit unsigned integer describing the size in bytes of the current SC3DMC stream.

encodingMode: A 8-bit unsigned integer indicating the encoding method for the connectivity

Table AMD1.2 — SC3DMC encoding modes

encodingMode	Method
0	QBCR
1	SVA
2	TFAN
3-255	ISO reserved

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.

numberOfNormal: A 32-bit unsigned integer indicating the number of normal coordinates.

numberOfTexCoord: A 32-bit unsigned integer indicating the number of texture coordinates.

numberOfColor: A 32-bit unsigned integer indicating the number of color coordinates.

numberOfOtherAttributes: A 32-bit unsigned integer indicating the number of the other attributes.

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.

numberOfNormalIndex: A 32-bit unsigned integer indicating the number of indices associated to the normals.

numberOfColorIndex: A 32-bit unsigned integer indicating the number of indices associated to the colors.

numberOfTexCoordIndex: A 32-bit unsigned integer indicating the number of indices associated to the texture coordinates.

numberOfOtherAttributesIndex: A 32-bit unsigned integer indicating the number of indices associated to the other attributes.

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.

TexCoordWidth: A 32-bit unsigned integer indicating the width size of texture image.

TexCoordHeight: A 32-bit unsigned integer indicating the height size of texture image.

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: A 32-bit floating point data indicating range value used for geometry quantization

quantRangeGeometryArr[]: 1 by 3 array containing 32 bit floating data indicating range value for each axes used for geometry quantization

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

quantRangeNormal: A 32-bit floating point data indicating range value used for normal quantization

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

quantRangeColor: A 32-bit floating point data indicating range value used for color quantization

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

quantRangeTexCoord: A 32-bit floating point data indicating range value used for texcoord quantization

quantRangeOtherAttributes: A 32-bit floating point indicating range value used for normal quantization

4.2.5.2.3 SC3DMCStreamData class

4.2.5.2.3.1 Syntax

```
class SC3DMCStreamData{
    if(encodingMode = 0)
    {
        DecodeIntArray (numberOfGeometry, numberOfCoordIndex*3, 1) decodedCoordIndex;
        if(numberOfNormalIndex != 0)
        {
            if (normalPerVertex == 1)
                DecodeIntArray (numberOfNormal, numberOfNormalIndex*3, 1) decodedNormalIndex;
            else
                DecodeIntArray (numberOfNormal, numberOfNormalIndex, 1)  decodedNormalIndex;
        }

        if(numberOfColorIndex != 0)
        {
            if (colorPerVertex == 1)
                DecodeIntArray (numberOfColor, numberOfColorIndex*3, 1)  decodedColorIndex;
            else
                DecodeIntArray (numberOfColor, numberOfColorIndex, 1)  decodedColorIndex;
        }
        If(numberOfTexCoord != 0)
        {
            DecodeIntArray (numberOfTexCoord, numberOfOTexCoordIndex*3, 1)
            decodedTexCoordIndex;
        }
        If(numberOfOtherAttributeIndex != 0)
        {
            if (otherAttributesPerVertex == 1)
                DecodeIntArray (numberOfotherAttributes, numberOfOtherAttributesIndex*
                dimensionOfOtherAttributes , 1)  decodedOtherAttributesIndex;
            else
                DecodeIntArray (numberOfOtherAttributesIndex, 1)  decodedOtherAttributesIndex;
        }
```

```
    If(numberOfCoord!= 0)
        DecodefloatArray (numberOfCoord, 3, quantMinGeometry, quantRangeGeometry, tQP)
  decodedCoord;

    If (numberOfNormal != 0)
        DecodefloatArray(numberOfNormal, 3, quantMinNormal, quantRangeNormal, tQP)
  decodedNormal;
    If (numberOfColor !=0)
        DecodefloatArray (numberOfColor, 3, quantMinColor, quantRangeColor, tQP)  decodedColor;
   If (numberOfTexCoord !=0)
        DecodefloatArray (numberOfTexCoord, 3, quantMinTexCoord, quantRangeTexCoord, tQP)
     decodedColor;

    If (numberOfOtherAttributes != 0)
        DecodefloatArray (numberOfOtherAttributes, dimensionOfOtherAttributes,
        quantMinOtherAttributes, quantRangeOtherAttributs, tQP)decodedOtherAttributes
  }
  else if (encodingMode = 1)
  {
      SVAIndexDecoder (numberOfGeometry, numberOfCoordIndex) decodedCoordIndex;

      if(numberOfNormalIndex != 0)
      {
          SVAIndexDecoder (numberOfNormal, numberOfNormalIndex) decodedNormalIndex;
       }

       if(numberOfColorIndex != 0)
      {
          SVAIndexDecoder (numberOfColor, numberOfColorIndex)  decodedColorIndex;
       }
      If(numberOfTexCoord != 0)
      {
          SVAIndexDecoder (numberOfTexCoord, numberOfTexCoordIndex)
          decodedTexCoordIndex;
       }
      If(numberOfOtherAttributeIndex != 0)
     {
       SVAIndexDecoder (numberOfotherAttributes, numberOfOtherAttributesIndex* dimensionOfOtherAttributes
, 1)  decodedOtherAttributesIndex;
      }

    If(numberOfCoord!= 0)
        DecodefloatArray (numberOfCoord, 3, quantMinGeometry, quantRangeGeometry, tQP)
  decodedCoord;

    If (numberOfNormal != 0)
        DecodefloatArray(numberOfNormal, 3, quantMinNormal, quantRangeNormal, tQP)
  decodedNormal;
    If (numberOfColor !=0)
        DecodefloatArray (numberOfColor, 3, quantMinColor, quantRangeColor, tQP)  decodedColor;
   If (numberOfTexCoord !=0)
        DecodefloatArray (numberOfTexCoord, 3, quantMinTexCoord, quantRangeTexCoord, tQP)
     decodedColor;

    If (numberOfOtherAttributes != 0)
        DecodefloatArray (numberOfOtherAttributes, dimensionOfOtherAttributes,
        quantMinOtherAttributes, quantRangeOtherAttributs, tQP)decodedOtherAttributes
  }
  else if (encodingMode = 2)
  {
```

```
        TFANIndexDecoder(3, numberOfCoord, numberOfCoordIndex, triangleOrderPres, 0)
        decodedCoordIndex;

        if(numberOfNormalIndex != 0)
        {
            if (normalPerVertex == 1)
                TFANIndexDecoder(3, numberOfNormal, numberOfNormalIndex, 1, 1)
                decodedNormalIndex;
    else
                SC3DMCDataDecoder(numberOfNormalIndex, 1)  decodedNormalIndex;
    }

    if(numberOfColorIndex != 0)
    {
            if (colorPerVertex == 1)
                TFANIndexDecoder(3, numberOfColor, numberOfColorIndex, 1, 1) decodedColorIndex;
            else
                SC3DMCDataDecoder(numberOfColorIndex, 1)  decodedColorIndex;
     }
     If(numberOfTexCoord != 0)
     {
             TFANIndexDecoder(3, numberOfTexCoord, numberOfOtherAttributesIndex, 1, 1)
             decodedTexCoordIndex;
     }
      If(numberOfOtherAttributeIndex != 0)
      {
            if (otherAttributesPerVertex == 1)
                TFANIndexDecoder(3, numberOfOtherAttributes, numberOfOtherAttributesIndex, 1, 1)
                decodedOtherAttributesIndex;
            else
                SC3DMCDataDecoder(numberOfOtherAttributesIndex, 1)  decodedOtherAttributesIndex;
      }
  If(numberOfCoord!= 0)
       DecodefloatArray (numberOfCoord, 3, quantMinGeometry, quantRangeGeometry, tQP)
  decodedCoord;

    If (numberOfNormal != 0)
            DecodefloatArray(numberOfNormal, 3, quantMinNormal, quantRangeNormal, tQP)
  decodedNormal;
    If (numberOfColor !=0)
            DecodefloatArray (numberOfColor, 3, quantMinColor, quantRangeColor, tQP)  decodedColor;
   If (numberOfTexCoord !=0)
            DecodefloatArray (numberOfTexCoord, 3, quantMinTexCoord, quantRangeTexCoord, tQP)
      decodedColor;

    If (numberOfOtherAttributes != 0)
            DecodefloatArray (numberOfOtherAttributes, dimensionOfOtherAttributes,
            quantMinOtherAttributes, quantRangeOtherAttributs, tQP)decodedOtherAttributes

  }
}
```


4.2.5.2.3.2 Semantics

decodedCoord: A reconstructed coordinate whose size is 1 by numberOfCoord*3.

decodedCoordIndex: A reconstructed coordIndex whose size is 1 by numberOfCoordIndex*3.

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

decodedNomalIndex: A reconstructed noramlIndex whose size is 1 by numberOfNormalIndex*3.

decodedTexCoord: A reconstructed texcoord whose size is 1 by numberOfTexCoord*2.

decodedTexCoordIndex: A reconstructed texcoordIndex whose size is 1 by numberOfTexCoordIndex*3.

decodedColor: A reconstructed color whose size is 1 by numberOfColor*3.

decodedColorIndex: A reconstructed colorIndex whose size is 1 by numberOfColorIndex*3.

decodedOtherAttributes: A reconstructed otherAttributes whose size is 1 by dimensionOfOtherAttributes*OfOtherAttributes.

decodedOtherAttributesIndex: A reconstructed otherAttributesIndex whose size is 1 by numberOfColorIndex*3.

tQP: 3 integer array containing data for quantization containing QP, width, and height values

DecodefloatArray: A decoding function for geometry, color, normal, and texture coordinate.

DecodeIntArray: A decoding function of Index depending on SC3DMC prediction mode for geometry, color, normal, and texture coordinate.

SVAIndexDecoder: A decoding function of index data using SVA algorithm.

TFANIndexDecoder: A decoding function of index data using TFAN algorithm.

4.2.5.2.4 DecodeFloatArray Class

4.2.5.2.4.1 Syntax

```
DecodefloatArray(numberOfDatal, dim, quantMin, quantRange, quantizationMode, tQP)
{
    DecodeIntArray ((1<<QP[0]), numberOfData, dim)
    InverseQauntization(quatizationMode, tQP, quantMin, quantRange)
}
```

4.2.5.2.4.2 Semantics

InverQuantization: A decoding process of inverse quantization depending on the quantization mode as follows:

Table AMD1.3 — Quantization modes

Quantization Mode	Quantization method
0	result = quantMin + (1<<QP)*quantRange*input

1	nbins = 1 << ((normal_quant - 3) / 2);
	y0 = nbins - ceil(sqrt(nbins*nbins - i));
	x0 = i + y0*y0;
	skew = (x0 & 1)*2.0/3.0;
	x0 = (x0 >> 1) & (nbins - 1);
	x = (float)x0 + skew;
	y = (float)y0 + skew;
	z = (float)nbins - x - y;
	n = 1.0/sqrt(x*x + y*y + z*z);
	x = (sx) ? -x*n : x*n;
	y = (sy) ? -y*n : y*n;
	z = (sz) ? -z*n : z*n;

2	x = texture_image_width *input y = texture_image_height *input

4.2.5.2.5 DecodeFloatArray Class

4.2.5.2.5.1 Syntax

```
DecodefloatArray(numberOfData, dim, quantMin, quantRange, quantizationMode, tQP)
{
    InverseBinarizeIntArray(numberOfdata, dim, binarizationMode);
    InversePredictIntArray(numberOfdata, dim, predictionMode, sizeOfData);
}
```

4.2.5.2.5.2 Sementics

InverseBinarizeIntArray: decoding process of inverse binarization depending on binarization mode

InversePredictIntArray: decoding process of inverse prediction depending on prediction mode

4.2.5.2.6 SC3DMCDataDecoder class

4.2.5.2.6.1 Syntax

```
InverseBinarizeIntArray (numberOfdata, dim)
{
    Bit(4) predictionMode;
    Bit(4) BinarizationMode;    -
    If ((binarizationMode == 0) && (predictionMode == 0))    // FL
    {
    Bit(8) QP;
        for(i=0;i< numberOfdata *dim;i++)
        {
            bit (QP) nData; // simple QBCR
        }
    }
    else if (binarizationMode == 1)// BPC
    {
    If(prediction = 1)
    If (predictionMode==3)      bit(1-7) predictor
        bit (5) prefixSize
        for(i=0;i< numberOfdata *dim;i++)
        {
            BPDecoder(prefix_size) nDifData
            If (predictionMode==1 || predictionMode==4)      bit(1) nSign
        }
    }
    else if  (binarizationMode == 2)      // 4C
    {
        for(i=0;i< numberOfdata *dim;i++)
        {
            If (predictionMode==3)
            {
                bit(3) predictor;
                bit (1) terminationBit
                while (terminationBit)
                {
                    bit(3)      threeBitsFL;
                bit(1)      terminationBit;
                }
            }
            else
            {
```