



INTERNATIONAL STANDARD ISO/IEC 14496-16:2004
TECHNICAL CORRIGENDUM 1

Published 2005-03-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Coding of audio-visual objects —
Part 16:
Animation Framework eXtension (AFX)

TECHNICAL CORRIGENDUM 1

Technologies de l'information — Codage des objets audiovisuels —
Partie 16: Extension du cadre d'animation (AFX)

RECTIFICATIF TECHNIQUE 1

Technical Corrigendum 1 to ISO/IEC 14496-16:2004 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

In Clause 3, replace the list with:

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

In subclause 4.3.1, remove reference to particle system:

- Particle systems, which consists of the following nodes: **Particles**, **ParticleInitBox**, **ParticleInitCone**, **ParticleObstacle**, and **PointAttractor**.

In subclause 4.3.4.1.3, replace:

A MeshGrid mesh allows view-dependent streaming (see subclause 5.1) and particular animation possibilities, such as: hierarchical grid animation, and offset based animation (see subclause 4.3.4.3).

with:

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

In subclause 4.4.2.1, replace Table 2: General arithmetic operator with:

Op	Description	Rules for each P(X1,X2,X3,X4)	Syntax
Sadd	Arithmetic addition of the density of two forms	$d_r(P) = d_0(P) + d_1(P)$	Sadd (F0, F1)
Smul	Arithmetic multiplication of the density of two forms	$d_r(P) = d_0(P) * d_1(P)$	Smul (F0, F1)
Sdif	The positive difference of two forms <i>F0</i> and <i>F1</i> .	Returns the difference between the densities if this result is nonnegative, and 0 if the result is negative.	Sdif (F0, F1)
Sexp	Exponentiation of forms	Raises one form to the power of another form. The density of a point with respect to <i>F1</i> serves as exponent to the density of the same point with respect to <i>F0</i> .	Sexp (F0, F1)
Sgcd	Greatest common divisor	Returns the <i>gcd</i> of the densities of two forms.	Sgcd (F0, F1)
Slcm	Least common multiple	Returns the <i>lcm</i> of the densities of two forms.	Slcm (F0, F1)
Smod	Integral remainder	The Integral remainder operator calculates the integer remainder when the density of a point with respect to <i>F0</i> is divided by the density of that point with respect to <i>F1</i> .	Smod (F0, F1)
Ssab	Absolute difference	Returns the result of the subtraction between the densities of two forms when this result is nonnegative. Otherwise, the return is the result opposed value.	Ssab (F0, F1)

Scub	Integral cube root	Returns the integral cube root of the density of the form.	Scub (F0)
Ssqr	Integral square root	Returns the integral square root of the density of the form.	Ssqr (F0)
Smax	Maximum	This operator is equivalent to the ternary union for n-ary logic.	Smax (F0, F1)
Smin	Minimum	This operator is equivalent to the ternary intersection for n-ary logic.	Smin (F0, F1)

In subclause 4.4.2.2, replace Table 3: Ternary logic operators with:

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

In subclause 4.4.2.3, replace Table 4: Filtering and test operators with:

Filter	Description	Rules	Syntax
Seqf	Equality filter	The density of the volume that checks the equality test, and 0 otherwise.	Seqf (F0, F1)
Sgtf	greater than or equal filter	The density of the second volume if the density of the first one is greater than or equal to the density of the second volume, and 0 otherwise.	Sgtf (F0, F1)
Sgtf	Greater than filter	The density of the second volume if the density of the first one is greater than the density of the second volume, and 0 otherwise.	Sgtf (F0, F1)
Sltef	Less than or equal filter	The density of the second volume if the density of the first one is less than or equal to the density of the second volume, and 0 otherwise.	Sltef (F0, F1)
Sltf	Less than filter	The density of the second volume if the density of the first one is less than the density of the second volume, and 0 otherwise.	Sltf (F0, F1)

Sevnf	Even filter	The volume density if the density is even, and 0 otherwise.	Sevnf (F0)
Soddf	Odd filter	The result is the volume density if the density is odd, and 0 otherwise.	Soddf (F0)
Sneqf	Difference filter	The result is the second volume density if the densities are different, and 0 otherwise.	Sneqf (F0, F1)

In subclause 4.4.3, replace old operators symbols by the new ones. Replace:

```
# b: egg cut by a box
Group{
  children [
    # Primary SolidRep
    DEF Le_Solid3 SolidRep{
      bboxSize 5 5 5
      densityList [1 3 4] # choice of densities to display
      dual FALSE
    }
    DEF Le_Script Script{ # Solid Tree definition
      field SFNode Srepout USE Le_Solid3
      #primitives
      field SFNode White # container
      Transform {
        translation 0 0 0
        children [
          Shape {
            ...
            geometry
            Quadric { P0 1 0 0 1
              P1 -1 0 0 1
              P2 0 1 0 0
              P3 0 0 1 0
              P4 0 1 0 1
              P5 0 0 1 1
              bboxSize 1 1.3 0.95
              solid FALSE}
          ]
        }
      }
      field SFNode Yolk # matter inside
      Transform {
        children [
          Shape {
            ...
            geometry
            Quadric { P0 1 0 0 1
              P1 -1 0 0 1
              P2 0 1 0 0
              P3 0 0 1 0
              P4 0 1 0 1
              P5 0 0 1 1
              bboxSize 0.7 0.7 0.4
              density 3
              solid FALSE}
          ]
        }
      }
      field SFNode CuttingBox # cutting tool
      Transform {
        translation 0.5 0 0
        children [
          Quadric { P0 -1 0 0 1
            P1 1 0 0 1
            P2 0 1 0 0
            P3 0 0 1 0
            P4 0 1 0 0
            P5 0 0 1 0
            bboxSize 1 1 1
            density 4
            solid FALSE}
          ]
        }
      }
    }
  ]
  directOutput TRUE
}
```

```

        url "vrmlscript:
        function initialize(){
            Srepout.solidTree = (White + Yolk)- CuttingBox;}
        "
    }
}

```

with:

```

# b: egg cut by a box
Group{
    children [
        # Primary SolidRep
        DEF Le_Solid3 SolidRep{
            bboxSize 5 5 5
            densityList [1 3 4] # choice of densities to display
            dual FALSE
        }
        DEF Le_Script Script{ # Solid Tree definition
            field SFNode Srepout USE Le_Solid3
            #primitives
            field SFNode White # container
            Transform {
                translation 0 0 0
                children [
                    Shape {
                        ...
                        geometry
                        Quadric { P0 1 0 0 1
                        P1 -1 0 0 1
                        P2 0 1 0 0
                        P3 0 0 1 0
                        P4 0 1 0 1
                        P5 0 0 1 1
                        bboxSize 1 1.3 0.95
                        solid FALSE}
                    ]
                }
            field SFNode Yolk # matter inside
            Transform {
                children [
                    Shape {
                        ....
                        geometry
                        Quadric { P0 1 0 0 1
                        P1 -1 0 0 1
                        P2 0 1 0 0
                        P3 0 0 1 0
                        P4 0 1 0 1
                        P5 0 0 1 1
                        bboxSize 0.7 0.7 0.4
                        density 3
                        solid FALSE}
                    ]
                }
            field SFNode CuttingBox # cutting tool
            Transform {
                translation 0.5 0 0
                children [
                    Quadric { P0 -1 0 0 1
                    P1 1 0 0 1
                    P2 0 1 0 0
                    P3 0 0 1 0
                    P4 0 1 0 0
                    P5 0 0 1 0
                    bboxSize 1 1 1
                    density 4
                    solid FALSE}
                ]
            }
            directOutput TRUE
            url "vrmlscript:
            function initialize(){

```

```

        "      Srepout.solidTree = Sdif(Sadd(White,Yolk), CuttingBox);}
    }
}

```

In subclause 4.5.1, replace:

Two technologies are defined in this subclause:

1. Depth image-based representation comprises the following nodes: **DepthImage**, **SimpleTexture**, **PointTexture**, and **OctreelImage**.
2. Light-field mapping comprises the following nodes: **LFM_Appearance**, **LFM_FrameList**, **LFM_LightMap**, **LFM_SurfaceMapList**, **LFM_ViewMapList**, and **LFM_BlendList**.

with:

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

In subclause 4.5.3, Light Field Mapping, remove the entire subclause.

In subclause 4.7.1.1.1, Node Interface, replace Types of the SBBone:

SBBone{ #%%NDT=SFSBBoneNode

with:

SBBone{ #%%NDT=SFSBBoneNode, SF3DNode, SF2DNode

In subclause 4.7.1.1.1, Node Interface, replace default value for the "scale" field:

exposedField SFVec3f scale 0 0 0

with:

exposedField SFVec3f scale 1 1 1

In subclause 4.7.1.2.1, Node Interface, replace Types of the SBSegment:

SBSegment{ #%%NDT=SFSBSegmentNode

with:

SBSegment{ #%%NDT=SFSBSegmentNode, SF3DNode, SF2DNode

In subclause 4.7.1.3.1, Node Interface, replace Types of the SBSite:

SBSite { #%NDT=SFSBSiteNode

with:

SBSite { #%NDT=SFSBSiteNode, SF3DNode, SF2DNode

In subclause 4.7.1.4.1, Node Interface, replace Types of the SBMuscle:

SBMuscle{ #%NDT=SFSBMuscleNode

with:

SBMuscle{ #%NDT=SFSBMuscleNode, SF3DNode, SF2DNode

In subclause 5.4.2.8.1, replace the definition of the bba_object_plane_mask class:

```
class bba_object_plane_mask() {
    bit(5) NumberOfInterpolatedFrames; //NIF
    if (isIntra){
        bit(5) bba_quant;
        bit(3) pow2quant;
        bit(10) NumberOfBones; //NSBB
        bit(10) NumberOfMuscles; //NSBM
        for (bone=1; bone<NumberOfBones; bone++){
            bit(10) BoneIdentifier; //IDB
            bone_mask bnmask();
        }
        for (ms=1; ms< NumberOfMuscles; ms++){
            bit(10) MuscleIdentifier; //IDM
            bit(6) NumberControlPoints; //NCP
            bit(6) NumberKnots; //NK
            muscle_mask msmask();
        }
    }
}
```

with:

```
class bba_object_plane_mask() {
    bit(5) NumberOfInterpolatedFrames; //NIF
    if (isIntra){
        bit(5) bba_quant;
        bit(3) pow2quant;
        bit(3) NumberOfControllerTypes;
        bit(10) NumberOfBones; //NSBB
        bit(10) NumberOfMuscles; //NSBM
        for (bone=1; bone<NumberOfBones; bone++){
            bit(10) BoneIdentifier; //IDB
            bone_mask bnmask();
        }
        for (ms=1; ms< NumberOfMuscles; ms++){
            bit(10) MuscleIdentifier; //IDM
            bit(6) NumberControlPoints; //NCP
            bit(6) NumberKnots; //NK
            muscle_mask msmask();
        }
    }
}
```

In subclause 5.4.2.8.2, add at the end of the subclause the following sentence:

NumberOfControllerTypes (NMF) - a 3-bit unsigned integer indicating the number of controller types used by the current version of the BBA stream. As this version of the BBA stream use bones and muscles this variable should take value « 2 » (two).

In subclause 5.4.2.9.1, replace:

```
class bone_mask() {
    bit(1) IsTranslation_changed;
    bit(1) marker_bit;
    if (IsTranslation_changed){
        bit(1) IsTranslationOnX_changed;
        bit(1) IsTranslationOnY_changed;
        bit(1) IsTranslationOnZ_changed;
    }
    bit(1) IsRotation_changed;
    bit(1) marker_bit;
    if (IsRotation_changed){
        bit(1) IsRotationOnAxis1_changed;
        bit(1) IsRotationOnAxis2_changed;
        bit(1) IsRotationOnAxis3_changed;
    }
    bit(1) IsScale_changed;
    bit(1) marker_bit;
    if (IsScale_changed){
        bit(1) IsScaleOnX_changed;
        bit(1) IsScaleOnY_changed;
        bit(1) IsScaleOnZ_changed;
    }
    bit(1) IsScaleOrientation_changed;
    bit(1) marker_bit;
    if (IsScaleOrientation_changed){
        bit(1) IsScaleOrientation AxisX_changed;
        bit(1) IsScaleOrientation AxisY_changed;
        bit(1) IsScaleOrientation AxisZ_changed;
        bit(1) IsScaleOrientation Value_changed;
    }
    bit(1) IsCenter_changed;
    bit(1) marker_bit;
    if (IsCenter_changed){
        bit(1) IsCenterOnX_changed;
        bit(1) IsCenterOnY_changed;
        bit(1) IsCenterOnZ_changed;
    }
}
```

with:

```
class bone_mask() {
    bit(1) IsTranslation_changed;
    bit(1) marker_bit;
    if (IsTranslation_changed){
        bit(1) IsTranslationOnX_changed;
        bit(1) IsTranslationOnY_changed;
        bit(1) IsTranslationOnZ_changed;
    }
    bit(1) IsRotation_changed;
    bit(1) marker_bit;
    if (IsRotation_changed){
        bit(1) isQuaternion ;
        if (!isQuaternion){
            bit(1) IsRotationOnAxis1_changed;
            bit(1) IsRotationOnAxis2_changed;
            bit(1) IsRotationOnAxis3_changed;
        }
    }
}
```

```

    }else{
    bit(1) IsRotationOnQx_changed;
    bit(1) IsRotationOnQy_changed;
    bit(1) IsRotationOnQz_changed;
    bit(1) IsRotationOnQw_changed;
    }
}
bit(1) IsScale_changed;
bit(1) marker_bit;
if (IsScale_changed){
    bit(1) IsScaleOnX_changed;
    bit(1) IsScaleOnY_changed;
    bit(1) IsScaleOnZ_changed;
}
bit(1) IsScaleOrientation_changed;
bit(1) marker_bit;
if (IsScaleOrientation_changed){
    bit(1) IsScaleOrientation AxisX_changed;
    bit(1) IsScaleOrientation AxisY_changed;
    bit(1) IsScaleOrientation AxisZ_changed;
    bit(1) IsScaleOrientation Value_changed;
}
bit(1) IsCenter_changed;
bit(1) marker_bit;
if (IsCenter_changed){
    bit(1) IsCenterOnX_changed;
    bit(1) IsCenterOnY_changed;
    bit(1) IsCenterOnZ_changed;
}
}
}

```

In subclause 5.4.2.12.2, replace:

NUM_SBCs=NSBB*16+NSBM*(NCP*3+NCP+NK)

with:

$$NUM_SBCs = \sum_{bn=1}^{NSBB} c_{bn} + \sum_{ms=1}^{NSBM} (3 * NCP_{ms} + NCP_{ms} + NK_{ms}),$$

with

$$c_{bn} = \begin{cases} 16, & \text{if } isQuaternion = 0 \\ 17, & \text{if } isQuaternion = 1 \end{cases}$$

is the number of the components for a bone transform.

In subclauses 5.1.1.2.1, Syntax of the Wavelet_Mesh_Object, replace:

```

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

```

with:

```

class Wavelet_Mesh_Object {
    int(1) isInBand;
    if (isInBand)

```

```

    WMDecoderConfig WMDecoderConfig;
    bit(1) WMOL;
    bit(1) isInLocalCoordinates;
    if (WMOL && WMDecoderConfig.hasScaleCoeff)
        Wavelet_Mesh_Object_Scale_Coeff Coefficients;
    ReadZT ZeroTree;
}

```

In subclause 5.2.1, replace:

The MeshGrid stream will, in general, consist of three parts: (1) a *connectivity-wireframe description*, (2) a *reference-grid description*, and (3) a *vertices-refinement description* (i.e. refining the position of the vertices relative to the reference-grid – the offsets). A minimal stream may, however, only consist of the description of the connectivity-wireframe, which is mandatory for every stream. All of these three parts can be encoded at each resolution level, either as one single *region of interest* (ROI) or in separate ROIs, if view-dependent decoding is needed.

with:

The MeshGrid stream has a modular structure consisting of a sequence of parts. There are several types of parts with different semantics, each one identified by a unique tag. All part types are optional, and several parts of the same type can be present in the stream, but their order is not imposed. The following parts of the MeshGrid stream are encoded at each resolution level, either as one single *region of interest* (ROI) or in separate ROIs, if view-dependent decoding is needed: (1) a *connectivity-wireframe description*, (2) a *reference-grid description*, and (3) a *vertices-refinement description* (i.e. refining the position of the vertices relative to the reference-grid – the offsets).

In subclause 5.2.2.1, Global Constants, remove:

```
const unsigned char MG_SIG = 0x10;    The signature of the MeshGrid stream
```

In subclause 5.2.2.1, Global Constants, replace:

```
READ_REPOSITION_BITS    No default value specified for the reposition bits
```

with:

```
const unsigned int READ_REPOSITION_BITS    No default value specified for the reposition bits
= 2
```

In subclause 5.2.2.2.1, Syntax for the MeshGridDecoderConfig, replace the entire section with:

```

class MeshGridDecoderConfig extends AFExtDescriptor: bit(8) tag=1{
    // the max number of resolution levels
    unsigned int totalNumLevelsMesh;

    // number of resolution levels specified for each u,v,w direction
    // of the reference-grid
    ParsableUVW nLevelsMesh(LEVEL_BITS);
    totalNumLevelsMesh = max(nLevelsMesh.u, max(nLevelsMesh.v, nLevelsMesh.w));

    // number of slices (reference surfaces) corresponding to the last
    // resolution level, in the u,v,w directions;
    ParsableUVW nSlices(SLICES_BITS);

    // flags

```

```

bit (1) hasConnectivityInfo;
bit (1) hasRefineInfo;
bit (1) hasRepositionInfo;
bit (1) hasGridInfo;

// reserved
bit(9) attributes = 0;

// type of homogeneous mesh
unsigned int(2) meshType;

if (meshType == QUADRI_MESH) {
    if (hasConnectivityInfo) {
        // connectivity bits and uniform quads splitting flags
        bit (1) sameBorderOrientation;
        bit (1) uniformSplit;
    }
    else {
        unsigned int sameBorderOrientation = 1;
        unsigned int uniformSplit = 1;
    }
}

if (nSlices.u == 1 || nSlices.v == 1 || nSlices.w == 1)
    GetValue offsetAmplitude;
}

// the choices for cyclic mesh
bit (3) cyclicMode;

// number of refine bits vertex
RefineVertexDescriptor refine;

// filter type for grid coding
unsigned int(FILTER_BITS) filterType;

// scale values for the x,y,z encoded grid coordinates
// minimum scale factor is MIN_SCALE
ScaleXYZ gridScale(FIELD_BITS);

// the grid corners
GridCorners gridCorner;
}

```

In subclause 5.2.2.2.2, Semantics for the MeshGridDecoderConfig, replace the entire section with:

The **MeshGridDecoderConfig** class initializes the MeshGrid decoder. It **(1)** parses the resolution levels of the mesh (*nLevelsMesh*) for the $\{u,v,w\}$ directions, encoded on LEVEL_BITS, with acceptable values for *nLevelsMesh.u*, *nLevelsMesh.v* and *nLevelsMesh.w*, lying in the range $[1, 2^{31}-1]$, **(2)** computes the maximum number of resolution levels *totalNumLevelsMesh*, **(3)** parses the number of slices (*nSlices*) $\{S_u, S_v, S_w\}$ corresponding to the last resolution level of the reference-grid, specified for the $\{u,v,w\}$ directions, with acceptable values for *nSlices.u*, *nSlices.v* and *nSlices.w*, lying in the range $[1, 2^{31}-1]$. It reads 5 values: **(4)** 1-bit flag *hasConnectivityInfo*, **(5)** 1-bit flag *hasRefineInfo*, **(6)** 1-bit flag *hasRepositionInfo*, **(7)** 1-bit flag *hasGridInfo*, and **(8)** 9-bit flag *attributes*.