INTERNATIONAL STANDARD

ISO/IEC 14496-16

Fourth edition
2011-11-01

AMENDMENT 1
2011-12-01

Information technology — Coding of audio-visual objects —

Part 16: Animation Framework eXtension (AFX)

AMENDMENT 1: Efficient representation of 3D meshes with multiple attributes

Technologies de l'information — Codage des objets audiovisuels —

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

AMENDEMENT 1: Représentation efficace de mailles 3D avec attributs multiples

Reference number
ISO/IEC 14496-16:2011/Amd.1:2011(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 1 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 1: Efficient representation of 3D meshes with multiple attributes

Add the following new 4.3.6 and renumber the curent 4.3.6 as 4.3.7:

4.3.6 Region-based representation of 3D meshes with multiple attributes

4.3.6.1 Introduction

The **IndexedRegionSet (IRS)** node specified in the next Subclause is based on the **IndexedFaceSet (IFS)** one but, thanks to its associated **Region** node, described as well below, it allows to group the faces of an IFS into subsets, or *regions*, to represent more naturally and code more efficiently 3D meshes of which some vertices have multiple attributes (colors, normal vectors or texture coordinates). The IRS concept is very adequate, for instance, to represent a mesh onto which several textures (one per mesh region) must be mapped. In this respect, IRS is somewhat related to the **MultiTexture[Coordinate]** nodes specified in 4.4.3, but IRS is more general, since it is not specifically designed for blending different textures in view-dependent rendering applications, nor exclusively targeted at specifying several texture coordinates per vertex. Indeed, it is for example also possible to model and code with very few bits, thanks to IRS, 3D objects such as a dice (i.e., a cube) with one solid color per square face as a triangle mesh with six regions of two triangles each. A similar result could be achieved with IFS, thanks to its per corner mapping of attributes, but this would be less efficient from a bitstream size viewpoint.

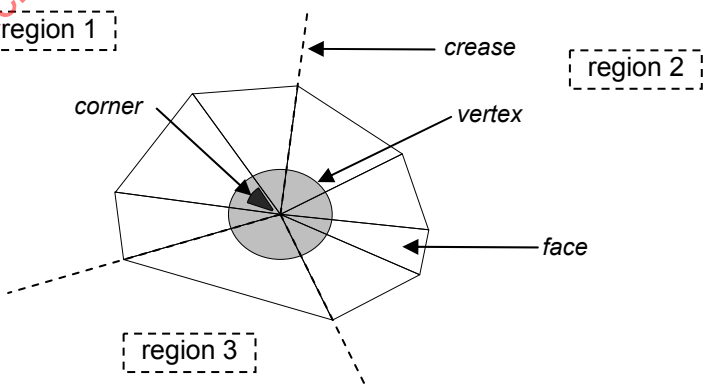

Figure Amd1.1 — A vertex shared by three regions

Figure Amd1.1 shows a triangle mesh vertex shared by eight triangles but only three regions, since just three of the edges incident to the vertex are part of region creases/frontiers. Region 1 contains three of the triangles shared by the vertex (and possibly others, not shown) whereas region 2 contains four and region 3 only one.

Defining mesh regions allows to specify attributes (not only texture coordinates) on a per *wedge* [39] basis, as well as to manage different texture images inside a single IRS node — note that this is impossible with the classic IFS node. The concept of wedge itself suffices to eliminate the redundancy implicit to replicating the attributes for different corners of the same vertex with the same attributes, and does so in the most versatile way possible. If slightly less versatile, the regions of IRS are more useful in practice since texture changes usually happen along edges.

A region is unambiguously defined by specifying a connected subset of the mesh triangles, the vertices included in that region being the ones that define the triangles in this subset. Only one attribute (e.g., one (*r*, *g*, *b*) color triplet or one (*u*, *v*) pair of texture coordinates) may be specified for each vertex inside a given region, but vertices sitting on region creases may have more attributes in their other region(s).

4.3.6.2 IndexedRegionSet node

4.3.6.2.1 Node interface

```
IndexedRegionSet {
    eventIn       MFInt32   set_colorIndex
    eventIn       MFInt32   set_coordIndex
    eventIn       MFInt32   set_normalIndex
    eventIn       MFInt32   set_texCoordIndex
    exposedField  SFNode    color           NULL
    exposedField  SFNode    coord           NULL
    exposedField  SFNode    normal          NULL
    exposedField  SFNode    texCoord        NULL
    field         SFBool    ccw             TRUE
    field         MFInt32   colorIndex      []      # [-1,inf)
    field         SFBool    colorPerVertex  TRUE
    field         SFBool    convex          TRUE
    field         MFNode    region          []
    field         SFFloat   creaseAngle     0       # [0,inf)
    field         MFInt32   normalIndex     []      # [-1,inf)
    field         SFBool    normalPerVertex TRUE
    field         SFBool    solid           TRUE
    field         MFInt32   texCoordIndex   []      # [0,inf)
}
```

4.3.6.2.2 Functionality and semantics

An **IRS** node has exactly the same fields as an **IFS** one, except for **coordIndex**, which has been replaced by **region**, meant to contain an array of **Region** nodes: faces originally described in the **coordIndex** field of the classic **IFS** node need to be classified into regions.

4.3.6.3 Region node

4.3.6.3.1 Node interface

```
Region {
    exposedField  SFNode    color           NULL
    exposedField  SFNode    normal          NULL
    exposedField  SFNode    texCoord        NULL
    exposedField  SFNode    texture         NULL
    exposedField  SFNode    textureTransform NULL
    field         MFInt32   colorIndex      []      # [-1,inf)
    field         MFInt32   coordIndex      []      # [-1,inf)
    field         MFInt32   normalIndex     []      # [-1,inf)
    field         MFInt32   texCoordIndex   []      # [0,inf)
}
```

4.3.6.3.2 Functionality and semantics

Since vertices may be shared between regions whereas faces may not, the **coord** field may only exist (i.e., be non-null) in the **IRS** node, while the **coordIndex** field may only exist (i.e., be non-empty) in the **Region** node.

As can be seen, the fields containing attributes or their corresponding indices coincide in the **IRS** and the **Region** nodes. The way attributes are specified depends on *whether* these fields exist or not; and, if they do, *where* they exist. There are two basic rules to avoid conflicts among them:

a) If one field exists inside the **IRS** node, its namesakes must not exist in any of its child **Region** nodes.

b) If one field (except for **texture[Transform]**) exists inside one **Region** node, it must also exist in all its sibling **Region** nodes, and not in its parent **IRS** node.

Let us analyze, for instance, the possible combinations of the **color[Index]** fields for the color attribute, which are summarized in the following table, where we use the adjectives “public” to refer to fields of the **IRS** node and “private” for the ones of the **Region** nodes.

		colorIndex		
		none	**one public**	**many private**
color	**none**	case N	not allowed	not allowed
	one public	case PO	case PI	case RPI
	many private	case RO	not allowed	case RI

Case N: No colors are specified.

Case PO: The color attribute does not follow the region division, and is applied once and only once to each vertex, regardless of the value of the **colorPerVertex** field. If there are *n* vertices in the **coord** field, then there shall be at least *n* colors in the **color** field.

Case PI: The color attribute does not follow the region division, and is applied once and only once to each vertex, regardless of the value of the **colorPerVertex** field. The **colorIndex** field shall contain at least as many indices as the number of vertices specified in **coord**. If the greatest index in **colorIndex** is *n*, then there shall be *n*+1 colors in the **color** field.

For the rest of the cases, the **colorPerVertex** field has to be taken into account. Its value being FALSE means that attributes are specified on a per face basis. When its value is TRUE, each vertex has as many attributes as regions it belongs to, and the implicit order of the vertices belonging to a region, used while explaining the correspondence between them and their attributes, is the one indicated by their position in **coord**.

Case RPI: In each **Region** node there is a **colorIndex** field whose indices refer to the **color** field in the parent **IRS** node. If the greatest index in **colorIndex** is *n*, there shall be at least *n*+1 colors specified in the public **color** field. If **colorPerVertex** is TRUE (respectively, FALSE), **colorIndex** shall contain at least as many indices as the number of different vertices (resp. faces) that there are in the region.

Case RO: In each **Region** node, one color from the local **color** field is assigned to each vertex or face of the region, depending on the value of the **colorPervertex** field. If **colorPerVertex** is TRUE (resp. FALSE), **color** shall contain at least as many colors as the number of different vertices (resp. faces) that there are in the region.

Case RI: In each **Region** node, the **colorIndex** field is used to assign each vertex a color from the **color** field in that same **Region** node. If the greatest index in **colorIndex** is *n*, there shall be at least *n*+1 colors specified the **color** field of its region. If **colorPerVertex** is TRUE (resp. FALSE), **colorIndex** shall contain at least as many indices as the number of different vertices (resp. faces) that there are in the region.

Normals are specified in exactly the same way as colors. For texture coordinates, however, there are some differences, the first being that the **texCoordPerVertex** field does not exist because it is assumed to always have the value TRUE. The **texture** and **textureTransform** fields in the **Region** node allow to specify different texture images for different regions, but the use of these fields is not mandatory. If texture coordinates are being specified in an **IRS** node, the texture image used for nodes with a null **texture** field shall be the one specified in the **appearance** field of the **Shape** node that the **IRS** belongs to. The same goes for the **textureTransform** field.

4.3.6.4 Examples

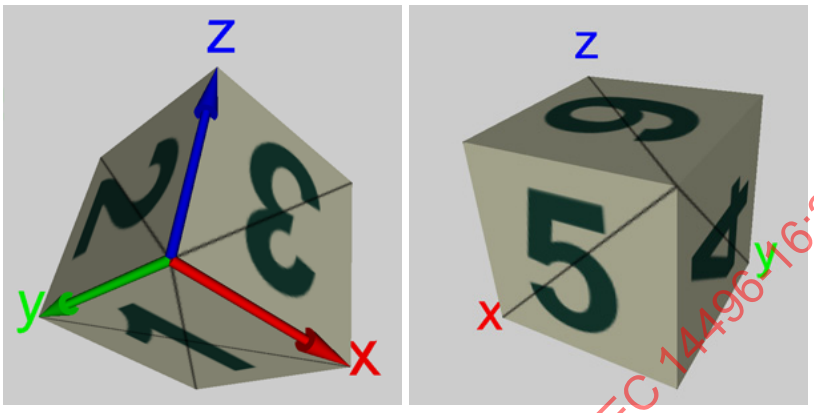

Figure Amd1.2 — Two views of a cube with nine vertices and six faces and texture regions

The following examples represent the dice shown in Figure Amd1.2 by means of **IRS** nodes, demonstrating the versatility of the IRS concept. All of them use the case RPI for texture coordinates, and have no color or normal attributes.

4.3.6.4.1 First IRS example

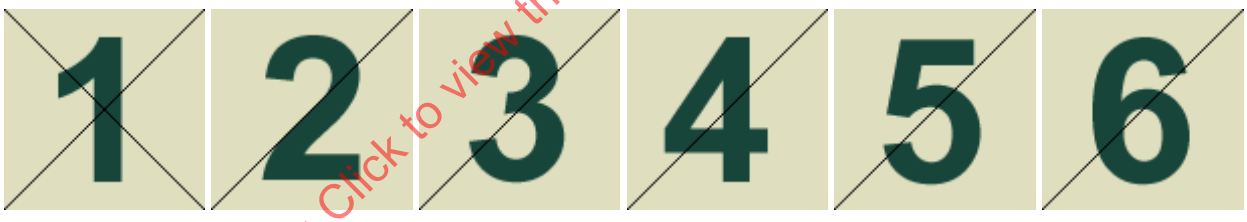

Figure Amd1.3 — The six images to be mapped on the cube above in the first IRS example

The first example uses the six images shown in Figure Amd1.3, one for each cube face. This results in six child Region nodes.

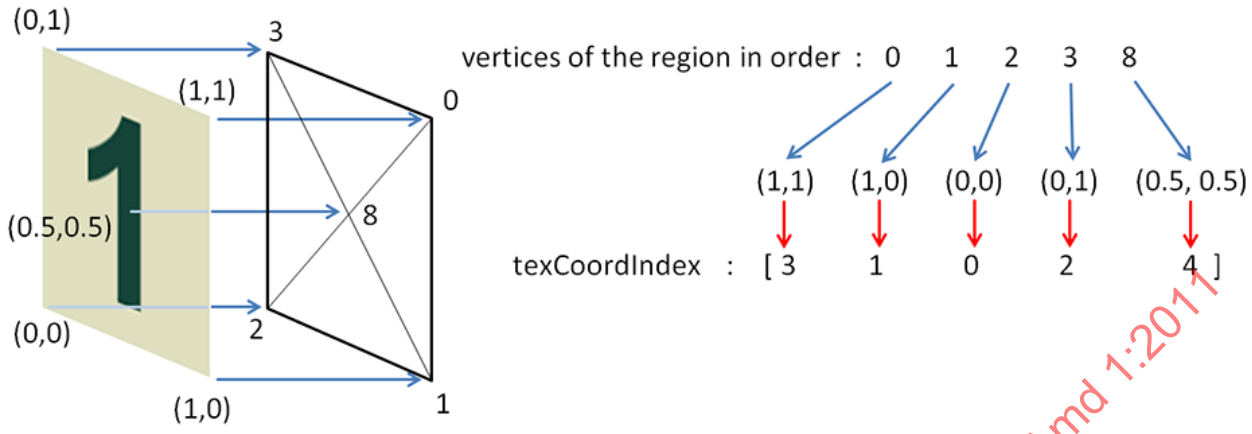

Figure Amd1.4 — Explanation of the texCoordIndex field for the first face/region of the first IRS example

```
IndexedRegionSet {
  coord Coordinate {point [
    0 0 0, 1 0 0, 1 1 0, 0 1 0, 0 1 1, 0 0 1, 1 0 1, 1 1 1, 0.5 0.5 0
  ]}
  texCoord TextureCoordinate {point [0 0, 1 0, 0 1, 1 1, 0.5 0.5]}
  region [
    Region {
      coordIndex [2 1 8 -1, 1 0 8 -1, 0 3 8 -1, 3 2 8]
      texCoordIndex [3 1 0 2 4]
      texture ImageTexture {url "../pix/1.png"}
    }
    Region {
      coordIndex [4 3 0 -1, 0 5 4]
      texCoordIndex [3 1 0 2]
      texture ImageTexture {url "../pix/2.png"}
    }
    Region {
      coordIndex [6 5 0 -1, 0 1 6]
      texCoordIndex [3 2 1 0]
      texture ImageTexture {url "../pix/3.png"}
    }
    Region {
      coordIndex [3 4 7 -1, 7 2 3]
      texCoordIndex [2 0 1 3]
      texture ImageTexture {url "../pix/4.png"}
    }
    Region {
      coordIndex [1 2 7 -1, 7 6 1]
      texCoordIndex [0 1 2 3]
      texture ImageTexture {url "../pix/5.png"}
    }
    Region {
      coordIndex [5 6 7 -1, 7 4 5]
      texCoordIndex [2 0 1 3]
      texture ImageTexture {url "../pix/6.png"}
    }
  ]
}
```

4.3.6.4.2 Second IRS example

The second example is based on the assumption that different texture images have to be used in different **Region** nodes, and that therefore the number of images to be wrapped around the cube can be decreased to two (shown in Figure Amd1.5), so that the **IRS** node can be encoded in a more compact way.