INTERNATIONAL STANDARD

ISO/IEC 14496-16

Second edition
2006-12-15

AMENDMENT 1
2007-10-01

Information technology — Coding of audio-visual objects —

Part 16: Animation Framework eXtension (AFX)

AMENDMENT 1: Geometry and shadow

Technologies de l'information — Codage des objets audiovisuels —

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

AMENDEMENT 1: Géométrie et ombre

Reference number
ISO/IEC 14496-16:2006/Amd.1:2007(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 14496-16:2006 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Coding of audio-visual objects —

Part 16: Animation Framework eXtension (AFX)

AMENDMENT 1: Geometry and shadow

Add subclause 4.3.6 MultiResolution FootPrint-Based Representation:

4.3.6 MultiResolution FootPrint-Based Representation

4.3.6.1 Introduction

MultiResolution FootPrint-Based representation is a solution to represent any set of objects based on footprints (a set of IndexedLineSet, or for a near future, buildings, cartoons…). The main interests in this representation are its progressivity, view dependency, and compression.

4.3.6.2 FootPrintSetNode

4.3.6.2.1 Node Interface

```
FootPrintSetNode { #NDT=%SFGeometryNode

    exposedField   MFGeometryNode        children          []
}
```

4.3.6.2.2 Functionality and semantics

The **children** field specifies the list of all footprints rendered according to the current viewpoint. This list contains currently FootPrintNode representing the set of footprints rendered from the current viewpoint. This list can be updated at each displacement of the viewpoint in order to adapt the scene complexity to the view. This representation can be extended to be used with any object based on footprints such as buildings, cartoons, etc. In this case, the children field can contain BuildingPartNode to represent buildings.

4.3.6.3 FootPrintNode

4.3.6.3.1 Node Interface

```
FootPrintNode { #NDT=%SFGeometryNode

    exposedField   SFInteger             index             -1
    exposedField   SFIndexLineSet2D      footprint         NULL
}
```

4.3.6.3.2 Functionality and semantics

Index: this is the index of the node corresponding to a footprint elevation at a specific level of detail. This index is essential for streaming, due to the synchronization between the representation on the server and on the client. This index will be sent to the server as a refinement request.

Footprint: this is an IndexLineSet2D describing the footprint.

4.3.6.4 BuildingPartPrintNode

4.3.6.4.1 Node Interface

```
BuildingPartNode { #NDT=%SFGeometryNode

    exposedField  SFInteger            index                -1
    exposedField  SFIndexLineSet2D     footprint            NULL
    exposedField  SFUnsigned integer   buildingIndex        -1
    exposedField  SFFloat              height               0
    exposedField  SFFloat              altitude             0
    exposedField  MFGeometryNode       alternativeGeometry  []
    exposedField  MFRoofNode           roofs                []
    exposedField  MFFacadeNode         facades              []
}
```

4.3.6.4.2 Functionality and semantics

Index: this is the index of the node corresponding to a footprint elevation at a specific level of detail. This index is essential for streaming, due to the synchronization between the representation on the server and on the client (this index will be sent to the server as a refinement request).

Footprint: this is a IndexLineSet2D describing the footprint.

buildingIndex: this is the index of the building to which this part is connected. A building corresponds to a group of building parts having the same buildingIndex.

Height: this is the height of the building.

Altitude: this is the altitude of the building (corresponding to the floor of the prism).

alternativeGeometry: this is a geometry node corresponding to an optional object used to replace the normal building. This alternative geometry can be used to swap a building with a more detailed model (used for example to replace a footprint elevation based model of a monument, by a more detailed model). In this case, the footprint-based elevation model will not be rendered, since the alternative model will be.

roofs: this is a node array allowing to describe complete roofs that will be reconstructed on top of the footprint elevation.

facades: this is a node array allowing to describe in detail the modelling of the façades corresponding to this building part. The size of this array corresponds to the number of facades, equivalent to the number of edges of the polygon defining the footprint.

4.3.6.5 RoofNode

4.3.6.5.1 Node Interface

```
RoofNode { #NDT=%SFGeometryNode

    exposedField  SFInteger       type              0
    exposedField  SFFloat         height            0.0
    exposedField  MFFloat         slopeAngle        [0.0]
    exposedField  SFFloat         eaveProjection    0.0
    exposedField  SFInt           edgeSupportIndex  -1
    exposedField  SFURL           roofTextureURL    ""
    exposedField  SFBool          isGenericTexture  TRUE
    exposedField  SFFloat         textureXScale     1.0
    exposedField  SFFloat         textureYScale     1.0
    exposedField  SFFloat         textureXPosition  0.0
    exposedField  SFFloat         textureYPosition  0.0
    exposedField  SFFloat         textureRotation   0.0
}
```

4.3.6.5.2 Functionality and semantics

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

height: this is the height of the roof that allows cropping it. (This is not used for flat roofs).

slopeAngle: this is the angle of the roof slopes in degrees (useless for flat roofs). In the case of a Symmetric Hip Roof, all slopes have the same angle. In the case of a Non Symmetric Hip Roof, each slope has a specific angle.

eaveProjection: this is the projection of the eave (useless for flat roofs).

edgeSupportIndex: this is the index of the edge in the footprint that supports the roof (use only for Salt Box roofs)

roofTextureURL: this is the URL of the texture that is orthogonally mapped onto the roof

isGenericTexture: this specifies whether the texture mapped onto the roof is generic or not. In the case of a generic texture, the reference system is centred on the top left vertex of the roof pan, and axed perpendicularly to the gutter. In the case of an aerial photograph, the reference system is centred on the first vertex of the footprint, and axed on the world coordinate system.

textureXScale: this is the scaling of the roof texture along X-axis

textureYScale: this is the scaling of the roof texture along Y-axis

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

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

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

4.3.6.6 FacadefNode

4.3.6.6.1 Node Interface

FacadeNode { #NDT=%SFGeometryNode
}

exposedField	**SFFloat**	WidthRatio	1.0
exposedField	**SFFloat**	XScale	1.0
exposedField	**SFFloat**	YScale	1.0
exposedField	**SFFloat**	XPosition	0.0
exposedField	**SFFloat**	YPosition	0.0
exposedField	**SFFloat**	XRepeatInterval	0.0
exposedField	**SFFloat**	YRepeatInterval	0.0
exposedField	**SFBool**	Repeat	FALSE
exposedField	**SFURL**	FacadePrimitive	""
exposedField	**SFInteger**	NbStories	0
exposedField	**MFInteger**	NbFacadeCellsByStorey	0
exposedField	**MFFloat**	StoreyHeight	1.0
exposedField	**MFFacadeNode**	FacadeCellsArray	[]

4.3.6.6.2 Functionality and semantics

WidthRatio: this corresponds to a ratio between the width of the cell compared to the width of the parent cells.

XScale: this is a parameter allowing scaling in X-coordinate the model corresponding to the URL Façade Primitive (2D texture or 3D model). For texture, this scale corresponds to the real size in X-coordinate in meters of the texture. For 3D Model, this size corresponds to the scale to apply on the model in X-coordinate.

NOTE This scale is very important as the model can be used for different buildings, and must be adjusted to the current one.

YScale is a parameter allowing scaling in Y-coordinate the model (2D texture or 3D model). For texture, this scale corresponds to the real size in Y-coordinate in meters of the texture. For 3D Model, this size corresponds to the scale to apply on the model in Y-coordinate.

NOTE This scale is very important as the model can be used for different buildings, and must be adjusted to the current one.

XPosition: this is a parameter allowing moving in X-coordinate the model in the cell defined by the FacadePrimitiveArray of the father node. This position can be essential to place a primitive in the centre of the cell.

YPosition: this is a parameter allowing moving in Y-coordinate the model in the cell defined by the FacadePrimitiveArray of the father node. This position can be essential to place a primitive in the center of the cell.

Repeat: this is a Boolean that is TRUE if and only if the model has to be repeated all over the cell defined by the father node.

NOTE This is essential for texture mapping, or to repeat regularly a model of windows all over a façade.

FacadePrimitive: this is a link to the corresponding primitive (Texture or 3D model) that have to be mapped onto the cell.

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

NbFacadeCellsByStorey: this is an array that defines the number of cells by storey. This parameter is essential to know on which storey corresponds a cell in FaçadeCellsArray.

StoriesHeight: this is an array specifying the height of each storey.

FacadeCellsArray: this is an array of FacadeNode that links each cell to a facadeNode (another array of cells, and/or a façade primitive like a texture or a 3D model). The size of this array is the sum of all NbFacadeCellsByStorey[i] , for all i from 0 to NbStories.

Add subclause 4.7.3 BBA Animation Algorithm:

4.7.3 BBA Animation algorithm

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

Animating the skinned model consists then in the following steps:

a) for all bones compute the initial transformation in the local space as the combination of the elementary transform: rotation, translation, center, scale and scaleOrientation; all these components are expressed in the parent's coordinate system.

b) for all the bones, compute the initial transformation in the world space as a product between the initial transformation of the bone in the local space and the initial transformation of the bone's parent expressed in the world space

c) compute the inverse of the previous transformation

d) at each animation frame, update the local elementary transforms: rotation, translation, center, scale and scaleOrientation

e) at each animation frame, repeat step b)

f) at each animation frame, for all the bones multiply the transformation obtained at step e) with the one computed at step c)

g) at each animation frame, for each pair bone/vertex, multiply the vertex with the transform obtained at the previous step and with the corresponding weight.

Add Subclause 4.8 Scene tools:

4.8 Scene tools

4.8.1 Shadows

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

4.8.1.1 Syntax

```
Shadow {
    eventIn        MFNode    addChildren
    eventIn        MFNode    removeChildren
    exposedField   MFNode    children              []
    exposedField   SFBool    enabled               TRUE
    exposedField   MFBool    cast                  TRUE
    exposedField    MFBool      receive                 TRUE
    exposedField    SFFloat     penumbra                0
}
```

4.8.1.2 Semantics

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

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

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

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

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

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

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

4.8.1.3 Annex

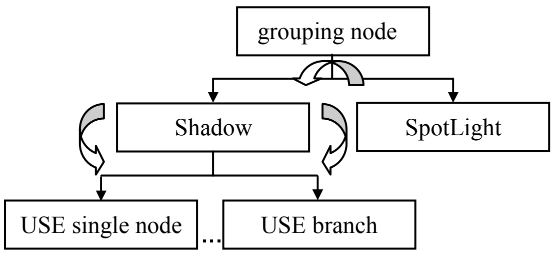

Figure AMD1-1 — Semantical representation

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

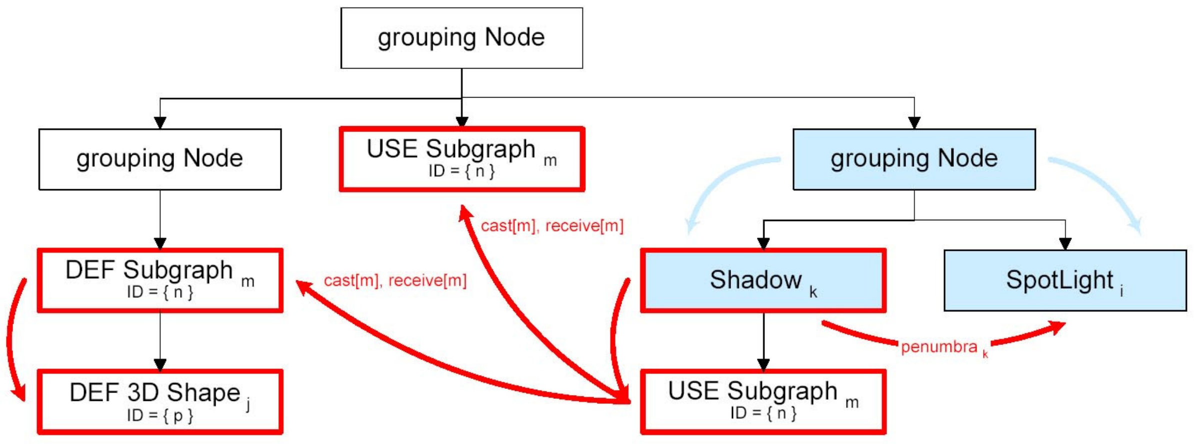

Figure AMD1-2 — Transmission of shadow properties to 3D-surfaces and light sources

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

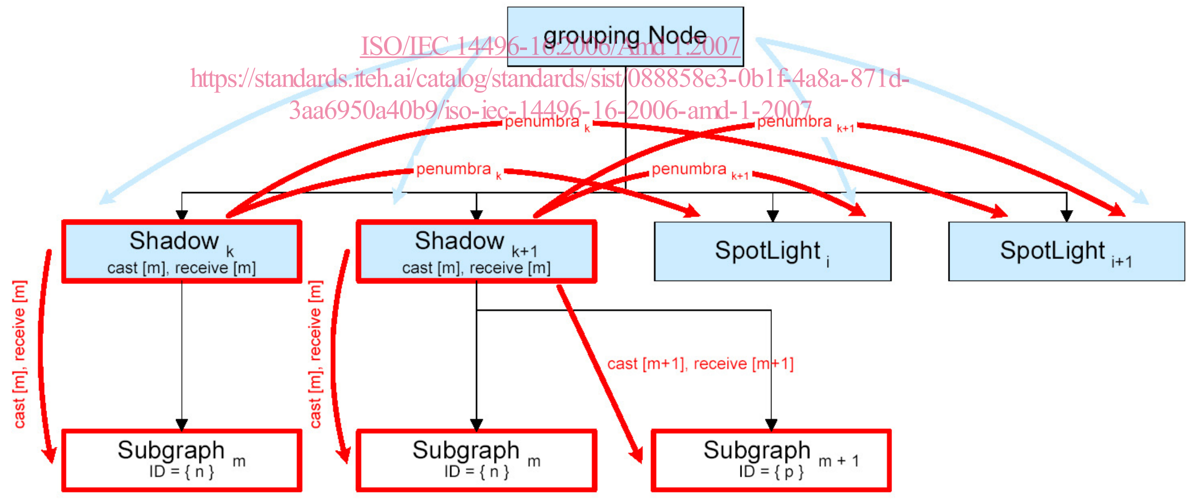

Figure AMD1-3 — Transmission of shadow properties to multiple 3D-surfaces and light sources

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

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

Add subclause 5.8 MultiResolution FootPrint-Based Representation:

5.8 MultiResolution FootPrint-Based Representation

5.8.1 Downstream syntax

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

5.8.1.1 FootPrintSetDecoderSpecificInfo

5.8.1.1.1 Syntax

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

5.8.1.1.2 Semantics

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

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

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

Step: this is the smallest spatial subdivision.

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

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

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

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

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

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

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

5.8.1.2 FPBuildingDecoderSpecificInfo

5.8.1.2.1 Syntax

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

5.8.1.2.2 Semantics

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

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

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

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

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

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

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

5.8.1.3 FootPrintSet Message

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

5.8.1.3.1 Syntax

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

5.8.1.3.2 semantics

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

5.8.1.4 FootPrint Message

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

5.8.1.4.1 Syntax

```
class FootprintMessage {
   int(FPNbBits) index
   bit(1) type
   FPNewVertices FPNV
   int(6) IndexNbBits
   if (type) {
      int(10) offspring
      for (i=0; i<offspring; i++) {
         int(FPNbBits) localIndex
         float(NbBitsMetricError) MetricError
         IndexFootprintSet IFPS
         switch (FPObjectType)
         {
            case 1: FPBuildingParameters FPBP
         }
      }
   }
   else
   {
      float(NbBitsMetricError) MetricError
      int(8) NbRings
      For (int i=0; i<NbRings-1; i++)
       {
         int(IndexNbBits) FirstVertexIndex
         switch (FPObjectType)
         {
            case 1: FPBuildingParameters FPBP
         }
       }
   }
}
```

5.8.1.4.2 Semantics

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

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

Index: this is the index identifying the current footprint.

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

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

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

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

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

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

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

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

5.8.1.5 FPNewVertices

5.8.1.5.1 Syntax

```
class FPNewVertices
{
   int(6) coordtype
   int(16) nbNewVertices
   for (i=0; i<nbnewvertices; i++)
   {
      if (type == 0 || step == -1.0)
      {
         float(32)     DeltaX
         float(32)     DeltaY
      }
      else
      {
         bool                        SignDeltaX
         unsigned int(coordtype-1)  AbsDeltaX
         bool                        SignDeltaY
         unsigned int(coordtype-1)  AbsdeltaY
      }
   }
}
```

5.8.1.5.2 Semantics

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

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

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

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

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

The decoding process is exposed in Annex J.

5.8.1.6 IndexFootprintSet

5.8.1.6.1 Syntax

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

5.8.1.6.2 Semantics

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

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