# VRML Extensions
# Supported by Cortona3D Viewer

# Table of Contents

## Introduction

The Visual Reality Modelling Language has a variety of powerful mechanisms that provide the content creators with almost unlimited capabilities of building 3D content. However, in certain cases, the implementation of the content creator's intent with the use of scene elements described in the VRML97 Specification can lead to a large file size, low performance of the VRML browser, etc. To broaden capabilities of content creators ParallelGraphics introduced support of a number of above elements as VRML extensions in Cortona3D Viewer. These extensions were implemented as new extension nodes or extended functionality of standard VRML nodes.

## Advanced Visual Effects

### CompositeTexture

The CompositeTexture3D and CompositeTexture2D nodes allow for adding composite textures to the 3D scene. You must have DirectX 9.0c installed and activated DirectX renderer with Auto or Concorde DX9 option in Cortona3D Viewer.

#### *CompositeTexture3D*

```
EXTERNPROTO CompositeTexture3D [
   eventIn   MFNode addChildren
   eventIn   MFNode removeChildren
   exposedField    MFNode children     NULL
   exposedField    SFInt32      pixelWidth   -1
   exposedField    SFInt32      pixelHeight  -1
   exposedField    SFBool repeatS      TRUE
   exposedField    SFBool repeatT      TRUE
   exposedField    SFNode background   NULL
   exposedField    SFNode fog          NULL
   exposedField    SFNode navigationInfo      NULL
   exposedField    SFNode viewpoint    NULL
]
[
   "urn:inet:parallelgraphics.com:cortona:CompositeTexture3D"
"http://www.cortona3d.com/source/extensions.wrl#CompositeTexture
3D"
]
```

The CompositeTexture3D node represents a texture mapped onto a 3D object that is composed of a 3D scene.

User interaction and the standard user navigation on the textured scene are disabled.

The *children* field is the list of 3D children nodes that define the 3D scene that forms the texture map.

The *addChildren* eventIn specifies a list of nodes that shall be added to the children field.

The *removeChildren* eventIn specifies a list of nodes that shall be removed from the children field.

The *pixelWidth* and *pixelHeight* fields specify the ideal size in pixels of this map. The default values result in an undefined size being used. This is a hint for the content creator to define the quality of the texture mapping.

The *background* field specifies the Background of the current texture. It may only contain Background node.

The *fog* field specifies the Fog node.

The *navigationInfo* field specifies the NavigationInfo node.The *viewpoint* field specifies the Viewpoint node.

The *repeatS* and *repeatT* fields specify how the texture wraps in the S and T directions.

## *CompositeTexture2D*

```
EXTERNPROTO CompositeTexture2D [
   eventIn   MFNode addChildren
   eventIn   MFNode removeChildren
   exposedField    MFNode children     NULL
   exposedField    SFInt32      pixelWidth   -1
   exposedField    SFInt32      pixelWidth   -1
   exposedField    SFBool repeatS      TRUE
   exposedField    SFBool repeatT      TRUE
   exposedField    SFNode background   NULL
   exposedField    SFNode viewport     NULL
]
[   "urn:inet:parallelgraphics.com:cortona:CompositeTexture2D"
"http://www.cortona3d.com/source/extensions.wrl#CompositeTexture
2D"
]
```

The CompositeTexture2D node represents a texture that is composed of a 2D scene, which may be mapped onto another object.

The *children* field contains a list of 2D children nodes that define the 2D scene that is to form the texture map.

The *addChildren* eventIn specifies a list of nodes that shall be added to the children field.

The *removeChildren* eventIn specifies a list of nodes that shall be removed from the children field.

The *pixelWidth* and *pixelHeight* fields specify the ideal size in pixels of this map. The default values result in an undefined size being used. This is a hint for the content creator to define the quality of the texture mapping.

The semantics of the *background* and *viewport* fields are identical to the semantics of the Layer2D fields of the same name.

The *repeatS* and *repeatT* fields specify how the texture wraps in the S and T directions.

## CubeEnvironment

The **CubeEnvironment** node specifies an environment cube texture map shape for simulating reflections on 3D objects in the scene.

```
EXTERNPROTO CubeEnvironment [
   exposedField  SFNode  backTexture
   exposedField  SFNode  bottomTexture
   exposedField  SFNode  frontTexture
   exposedField  SFNode  leftTexture
   exposedField  SFNode  rightTexture
   exposedField  SFNode  topTexture
]
[
   "urn:inet:parallelgraphics.com:cortona:CubeEnvironment"
"http://www.cortona3d.com/source/extensions.wrl#CubeEnvironment"
]
```

Fields and events:

- **Back Texture**: specifies the back texture in the cube map shape;
- **Bottom Texture:** specifies the bottom texture in the cube map shape;
- **Front Texture:** specifies the front texture in the cube map shape;
- **Left Texture:** specifies the left texture in the cube map shape;
- **Right Texture:** specifies the right texture in the cube map shape;
- **Top Texture:** specifies the right texture in the cube map shape.

Following nodes can be used as values of the above fields: ImageTexture, PixelTexture, MovieTexture, BumpMap, and MipMap. It is not possible to use the BumpMap or MipMap nodes as values of the fields if the CubeEnvironment node is in its turn specified in one of these two nodes.

Important: This node is supported by most video cards when the DirectX Renderer (Concorde DX7) or OpenGL Renderer is selected. It is strongly recommended to install the latest version of DirectX and update the video card driver (from its manufacturer's Website) before viewing VRML scenes containing this node.

## SphereEnvironment

The **SphereEnvironment** node specifies a spherical environment map for simulating reflections on 3D objects in the scene.

```
EXTERNPROTO SphereEnvironment [
     exposedField   SFNode  texture
]
[
   "urn:inet:parallelgraphics.com:cortona:SphereEnvironment"
   "http://www.cortona3d.com/source/extensions.wrl#SphereEnviron
   ment"
]
```

Fields and events:

- **Texture**: specifies texture of a reflecting shape. Can be ImageTexture, PixelTexture, MovieTexture, BumpMap, or MipMap node. It is not possible to use the BumpMap or MipMap nodes as values of the fields if the SphereEnvironment node is in its turn specified in one of these two nodes.

During the viewer's motion around the 3D object distortions or other artifacts in the reflection can take place. These problems can be resolved with the use of the cube environmental mapping.

## AdvancedAppearance

The AdvancedAppearance node enables you to use advanced texturing techniques such as multiple texturing.

## Node description

```
AdvancedAppearance {
     exposedField   SFNode      material            NULL
     exposedField   MFNode      textures            []
     exposedField   MFString    mappingTypes []
     exposedField   MFFloat     weights             []     # [0,
inf)
     exposedField   SFFloat     materialBlending    0      # [0,
1]
     exposedField   MFNode      textureTransforms   []
     exposedField   MFString    backgroundFactor    []
     exposedField   MFString    foregroundFactor    []
}
```

- The **material** field, if specified, contains a **Material** node.

- The **textures** field specifies a set of 2D textures for multi-texturing. The texture field, if specified, contains one of the various types of texture nodes (ImageTexture, MovieTexture, or PixelTexture). If the texture node is NULL or the texture field is unspecified, the object that references this Appearance is not textured.

- The **mappingTypes** field defines a texture map type. The possible types are:
  "SIMPLE" - ordinary mapping that all VRML browsers support,
  "ENVIRONMENT" - this simulates the reflecting surfaces.

  When there are several textures with the "SIMPLE" map type, it is possible to define individual mapping for each texture by using several sets of texture indexes in the **texCoordIndex** field of the geometry node. The **texCoordIndex** field may contain **N** * **L** indexes, where **N** is the number of textures with the "SIMPLE" map type, **L** - the number of indexes in the **coordIndex** field.
  For textures with the "ENVIRONMENT" map type texture indexes are not used.

- The **weights** field specifies a set of weights that are required to mix different textures. In Cortona 4.0 this field is ignored and the **backgroundFactor** and **foregroundFactor** fields are used for mixing textures.

- The **materialBlending** field specifies how to combine textures and materials on associated geometry. The value of **materialBlending** ranges from 0 to 1. If you don't specify any textures (the textures field is empty) or material (material is NULL), no combination happens. The weights required for mixing can be computed as:

```
if(textures.count == 0)
       Wmaterial = 1;      // a material is used
```

```
    else if(material == NULL)
          Wmaterial = 0;      // textures are used
    else          // textures and material are combined
          Wmaterial = materialBlending.
```

- The **textureTransforms** field specifies a set of 2D transformations that are applied to different textures that are specified in the textures field. The field, if specified, contains a list of TextureTransform nodes. Descriptions of the TextureTransform node are provided in the VRML97 specification (see 6.49, TextureTransform).The **backgroundFactor** and **foreGroundFactor** fields specify the sets of factors required to mix textures.Multi-texturing is implemented by the multi-pass rendering. On each pass the successive texture from the **textures** node is mixed with the color resulting from the previous pass according to the following formula:

$\mathbf{C_{MT}(i) = C_B * F_B(i) + C_F * F_F(i)}$, where

$\mathbf{C_{MT}(i)}$ - the color resulting from the given pass,
$\mathbf{i}$ - the texture number in the textures node (the pass number),
$\mathbf{C_B}$ - the pixel color resulting from the previous pass ($C_{MT}$(i - 1)), or back color (the frame-buffer contents before rendering) for the first pass,
$\mathbf{C_F}$ - the pixel color of the texture with the number i,
$\mathbf{F_B(i)}$ - the factor defined in the backgroundFactor field,
$\mathbf{F_F(i)}$ - the factor defined in the foregroundFactor field.

All colors are considered to have four components (RGBA). If there is no alpha channel (for example, the texture or back color has no alpha), the alpha value is considered to be 1 (entirely nontransparent color), i.e. the alpha channel takes part in all calculations equally with the other color components. The **backgroundFactor** and **foreGroundFactor** fields can take on the following values:

| **Value** | **Factor** |
|---|---|
| DEFAULT | default |
| ZERO | 0 |
| ONE | 1 |
| FORE_COLOR | $C_F$ |
| INV_FORE_COLOR | $1 - C_F$ |
| FORE_ALPHA | $A_F$ |
| INV_FORE_ALPHA | $1 - A_F$ |
| BACK_COLOR | $C_B$ |
| INV_BACK_COLOR | $1 - C_B$ |
| BACK_ALPHA | $A_B$ |
| INV_BACK_ALPHA | $1 - A_B$ |

$\mathbf{A_F}$ - the alpha value resulting from the given pass,
$\mathbf{A_B}$ - the alpha value resulting from the previous pass or from the back color. (If there is no alpha channel $\mathbf{A_B}$ = 1).

Only GeForce adapters support the last two values. So for the rest display adapters the **"BACK_ALPHA"** value is equivalent to **"ONE"**, and **"INV_BACK_ALPHA"** is equivalent to **"ZERO"**.

If the values in the **backgroundFactor** and **foreGroundFactor** fields are not defined or set to **"DEFAULT"**, they are determined by default according to the texture type (see the table below). The value of $\mathbf{C_F}$ also depends on the texture type. If the material is defined (lighting on), the intensity textures are modulating by the diffuse color of the material. If all textures have alpha channels, the material transparency is ignored i.e. it is considered to be 0, and alpha from the textures is used. But if there is at least one texture without alpha the material transparency modulates the alpha channel of all textures.

| **Texture type** | **backgroundFactor by default** | **foregroundFactor by default** | $\mathbf{C_F}$ **lighting on** | $\mathbf{C_F}$ **lighting off** |
|---|---|---|---|---|
| Intensity | "ZERO" | "ONE" | $D_M * C_T$ | $C_T$ |
| Intensity + Alpha | "INV_FORE_ALPHA" | "FORE_ALPHA" | $D_M * C_T$ | $C_T$ |
| RGB | "ZERO" | "ONE" | $C_T$ | $C_T$ |
| RGB + Alpha | "INV_FORE_ALPHA" | "FORE_ALPHA" | $C_T$ | $C_T$ |

$\mathbf{C_T}$ - is the color of the texture (all components are equal for intensity textures)
$\mathbf{D_M}$ - is the diffuse color of the material (the **Material.diffuseColor** field).
For compatibility with the standard specification, the following is implemented: if all textures have alpha channels, the material transparency is ignored i.e. it is considered to be 0, and the alpha from the textures is used. But if there is at least one texture without alpha the material transparency modulates the alpha channel of all textures. The AF value is calculated according to the formula:

```
 AF = AT * (1 - TM), where
      AT    - the alpha value of the texture
                   (for 1- and 3-component textures AT = 1)
      TM    - the Material.transparency field value
                   (or 0 if all textures have the alpha channel)
```

The color obtained after multi-texturing $\mathbf{C_{MT}}$ is processed in the following way:

**Lighting off**
The resulting color is calculated by the formula:
$\mathbf{C = B_M * E_M + (1 - B_M) * C_{MT}}$, where
**C** - is the resulting color of the pixel,
$\mathbf{B_M}$ - is the coefficient of the material blending (the **materialBlending** field),
$\mathbf{E_M}$ - is the emissive color of the material (the **Material.emissiveColor** field).

**Lighting on**

The resulting color of the pixel is calculated by the standard formula of the VRML lighting model where a diffuse factor ($\mathbf{O_{Drgb}}$) is set to the following color:
$\mathbf{O_{Drgb} = B_M * D_M + (1 - B_M) * C_{MT}}$

The resulting color **Ñ** is combined with the color resulting from the previous pass (which is equal to $\mathbf{C_{MT}}$) and with the transparency of Material (the **Material.transparency** field) according to the following formula:

$\mathbf{(1 - T_M) * C + T_M * C_{MT}}$

## TextureTransform

The **TextureTransform3** node defines a 3D transformation that is applied to texture coordinates used in environment mapping (3D transformation of the reflection vector).

```
EXTERNPROTO    TextureTransform3 [
   exposedField SFVec3f     center             0 0 0    # (-
inf,inf)
   exposedField SFRotation  rotation           0 0 1 0  # [-1
1],(-inf,inf)
   exposedField SFVec3f     scale              1 1 1    #
(0,inf)
   exposedField SFRotation  scaleOrientation   0 0 1 0  # [-1
1],(-inf,inf)
   exposedField SFVec3f     translation        0 0 0    # (-
inf,inf)
]
[
   "urn:inet:parallelgraphics.com:cortona:TextureTransform3"
"http://www.cortona3d.com/source/extensions.wrl#TextureTransform
3"
]
```

The fields of the TextureTransform3 node are analogous to the corresponding fields of the Transforms VRML node.

The TextureTransform3 node can be used as a value of the textureTransforms field of the AdvancedAppearance node. The texture, to which the transformation applies, should be specified in the textures field of the AdvancedAppearance node by the CubeEnvironment or SphereEnvironment nodes.

## SFVec2f Interpolator

The **Position2Interpolator** node linearly interpolates among a list of 2D vectors. This node allows a dynamic transformation that is applied to texture coordinates without implementation of the Script node.
Node description:

```
Position2Interpolator {
   eventIn          SFFloat set_fraction
   exposedField     MFFloat key         []
   exposedField     MFVec2f keyValue          []
   eventOut         SFVec2f value_changed
}
```

All definitions of the fields are similar to the VRML97 definitions of the **PositionInterpolator** node.

**Example:**

```
#VRML V2.0 utf8
NavigationInfo {
  type "EXAMINE"
}
Transform {
  rotation 1 1 1 1
  children [
   Shape {
    geometry Box {}
    appearance Appearance {
     texture ImageTexture {
      url "sky01.gif"
}
     textureTransform
      DEF TT TextureTransform {}
    }
   }
  ]
}
DEF TIS TimeSensor {
  loop TRUE
  cycleInterval 5
}
DEF PI2 Position2Interpolator {
  key [0 1]
  keyValue [0 0, 1 1]
}
ROUTE TIS.fraction_changed
TO PI2.set_fraction
ROUTE PI2.value_changed
TO TT.translation
```

## GradientBackground

The GradientBackground allows for creating horizontal or vertical gradient background that is static relatively to the camera movements.

```
EXTERNPROTO GradientBackground [
   eventIn         SFBool       set_bind
   exposedField          MFColor            color        0,0,0
   exposedField          MFFloat            colorPosition      0
   exposedField          SFString     type         "LINEAR-
VERTICAL"
   eventOut        SFBool       isBound
]
[
   "urn:inet:parallelgraphics.com:cortona:GradientBackground"
"http://www.cortona3d.com/source/extensions.wrl#GradientBackgrou
nd"
]
```

The *set_bind* field works in the same way as the set_bind field of the Background node.
The *color* field specifies two or more colors of the gradient.
The *colorPosition* field specifies the positions of colors listed in the color field. If only two colors are used to create the gradient, the colorPosition field is not necessary. If the value of the colorPosition field is not specified, the colors are arranged uniformly.
The *type* field specifies whether the gradient is horizontal or vertical. The possible values are "LINEAR-VERTICAL" (default) and "LINEAR-HORIZONTAL".
The *isBound* field works in the same way as the isBound field of the Background node.

# Geometry

## Splines

### *Cortona spline technology*

ParallelGraphics developed spline representation of geometry objects. Typically, by using VRML97 file format, the faces are used to build curvy shapes. There are two choices to avoid faceted shading: use many more faces to approximate the smooth shape, or shade the faces differently so it looks like you used lots of faces (Gouraud method). However, the spline objects are geometrically smooth and allow to transform them to any required quantity of faces to display.

The key benefits from the use of spline objects include:

- The high quality of a smooth surface can be described using VRML.
- It is required less computing resources for achievement of high quality.
- The size of the VRML-file decreases essentially (up to 10 times) without degradation of surface quality.
- The smooth variation of input parameters results in transformation of the spline surface that is worth to simplify the creation of realistic animations.
- The dynamic detailed elaboration provides the balance between necessary quality and frame-rate.
- The function of quality allows to operate a degree of detailed elaboration of a surface.

ParallelGraphics offers six new nodes:

- SplineCone
- SplineCylinder
- SplineElevationGrid
- SplineExtrusion
- SplineFaceSet
- SplineSphere

They are based on the standard nodes and expressed in using the VRML prototyping mechanism (external prototypes). Each new node contains both the same fields standard VRML node includes and extensions controlling level of detailed elaboration of a surface. ParallelGraphics' Cortona VRML client supports the proposed nodes and uses the spline representation for objects. The simplest way to use spline nodes is that of changing the appropriate node name. Other browsers interprets these nodes as Cone, Cylinder, ElevationGrid, Extrusion, FaceSet, and Sphere.

The method of spline representation of geometry objects is based on three-cubic spline interpolation, which is performed with Cortona VRML client automatically using incoming polygonal data (control vertices). So a spline surface includes all control vertices of the polygonal model. While rendering, a surface is broken down (tessellated) into a set of triangles approximating the spline surface. To balance between quality and the frame-rate, a surface curvature and number of triangles are taken into account. Moreover, it's possible to control the tessellation using the quality function.

ParallelGraphics has also developed the suitable converter for the translation of standard polygonal objects to the spline analogies.

## *Geometric spline nodes*

Each node corresponds to standard VRML node except the fields distance and quality. All other field definitions are similar to the VRML97 Node Reference. This section provides a detailed definition of the syntax of proposed nodes.

```
SplineCone {
   field     SFFloat     bottomRadius 1
   field   SFFloat   height       2
   field   SFBool    side         TRUE
   field   SFBool    bottom       TRUE
   field   MFFloat   distance     10
   field   MFFloat   quality      [0, 0.75]
}

SplineCylinder {
   field          SFBool bottom TRUE
   field          SFFloat     height 2
   field          SFFloat     radius 1
   field          SFBool side   TRUE
   field          SFBool top    TRUE
   exposedField   MFFloat     distance 10
   exposedField   MFFloat     quality  [0, 0.75]
}

SplineElevationGrid {
   eventIn        MFFloat     set_height
   exposedField   SFNode color        NULL
   exposedField   SFNode normal       NULL
   exposedField   SFNode texCoord     NULL
   field          MFFloat     height        []
   field          SFBool ccw          TRUE
   field          SFBool colorPerVertex     TRUE
   field          SFFloat     creaseAngle   0
   field          SFBool normalPerVertex    TRUE
   field          SFBool solid        TRUE
   field          SFInt32     xDimension    0
   field          SFFloat     xSpacing            0.0
   field          SFInt32     zDimension    0
   field          SFFloat     zSpacing            0.0
   exposedField   MFFloat     distance            10
   exposedField   MFFloat     quality             [0, 0.75]
}

SplineExtrusion {
   eventIn        MFVec2f     set_crossSection
   eventIn        MFRotation  set_orientation
   eventIn        MFVec2f     set_scale
   eventIn        MFVec3f     set_spine
   field          SFBool      beginCap      TRUE
   field          SFBool      ccw           TRUE
```

```
    field           SFBool      convex      TRUE
    field           SFFloat     creaseAngle  0
    field           MFVec2f           crossSection [1 1,
                                               1 -1,
                                               -1 -1,
                                               -1 1,
                                               1 1]
    field           SFBool      endCap      TRUE
    field           MFRotation  orientation 0 0 1 0
    field           MFVec2f     scale       1 1
    field           SFBool      solid       TRUE
    field           MFVec3f           spine        [0 0 0,
                                                    0 1 0]
    field           MFFloat     distance            10
    field           MFFloat     quality             [0, 0.75]
}

SplineFaceSet {
    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         []
    field           SFBool colorPerVertex   TRUE
    field           SFBool convex           TRUE
    field           MFInt32     coordIndex         []
    field           SFFloat     creaseAngle        0
    field           MFInt32     normalIndex        []
    field           SFBool normalPerVertex  TRUE
    field           SFBool solid            TRUE
    field           MFInt32     texCoordIndex      []
    exposedField    MFFloat     distance            10
    exposedField    MFFloat     quality             [0, 0.75]
}

SplineSphere {
    field           SFFloat     radius     1
    exposedField    MFFloat     distance   10
    exposedField    MFFloat     quality    [0, 0.75]
}
```

The smoothing surfaces are available only if the field normalPerVertex is set to TRUE (default value) that corresponds to Gouraud method for polygonal objects. If a set of 3D surface normal vectors is defined in node (the normal field), Cortona will use it in generating of a spline surface. If the normal field is NULL, the browser treats the normals automatically generated, using the creaseAngle field (see VRML97, Node Reference). The resulted spline surface contains all of vertices defined in node, and normals per vertex coincide with normals per spline surface at vertices.

The distance and quality fields enable to control the quality of the spline surface breakdown (triangulation) depending on the distance from the camera to the center of the object bounding box. The distance field specifies a set of distances to object. Each distance[i] value corresponds to the quality[i+1] value. The quality[0] value specifies the triangulation quality at distance=0. Thus, if the greatest index in the distance field is N, there shall be N+1 qualities in the quality field.

Let N denotes the greatest index in the distance field. The following equations define the current quality of the spline surface breakdown:

```
D <= distance[0]:

Q = quality[0] + (quality[1] - quality[0]) * D / distance[0]

distance[i] < D < distance[i+1]:

Q = quality[i+1] + (quality[i+2] - quality[i+1]) * (D -
distance[i]) / (distance[i+1] - distance[i])

distance[N-1] < D:

Q = quality[N],
```

where:
D - the distance from the camera to the center of the object bounding box.
Q - the current quality of the spline surface breakdown, ranging from 0 for the worst quality to 1 for best surface.

## NURBS

### *NURBS in Cortona*

Extending VRML standard ParallelGraphics implemented support for a mathematical model for surfaces known as NURBS for Cortona VRML client 1.5+. With NURBS geometry, users can model complex sculptured shapes faster, more accurately, and with fewer surfaces.

For example, NURBS geometry makes it possible to treat the hood of an automobile or the wing of an airplane as a single surface and create more realistic shapes of human bodies.

With NURBS it is easier to create virtual worlds in VRML with smooth surfaces and reduced download size of VRML files because of the compact NURBS description at once.

## *Geometric NURBS node*

This description corresponds to NURBS Extension for VRML97 Discussion & Node proposal 12 March, 1999 by Blaxxun interactive except the fields distance, quality, uTessellation, vTessellation, and texCoord.

```
NurbsSurface {
   field          SFInt32 uDimension 0      #[0, inf)
   field          SFInt32 vDimension 0      #[0, inf)
   field          MFFloat uKnot []          #(-inf,inf)
   field          MFFloat vKnot []          #[2, inf)
   field          SFInt32 uOrder 3          #[2, inf)
   field          SFInt32 vOrder 3          #[2, inf)
   exposedField   MFVec3f controlPoint []   #(-inf,inf)
   exposedField   MFFloat weight []         #(0, inf)
   exposedField   SFInt32 uTessellation 0   #(-inf,inf)
   exposedField   SFInt32 vTessellation 0   #(-inf,inf)
   exposedField   SFNode texCoord []
   exposedField   SFBool ccw TRUE
   exposedField   SFBool solid TRUE
   exposedField   MFFloat distance 10
   exposedField   MFFloat quality [0, 0.75]
}
```

- **uDimension** and **vDimension** define the number of control points in the u and v dimensions.

- **uOrder** and **vOrder** define the order of surface. From a mathematical point of view, the surface is defined by polynomials of the degree order-1.

  The order of the curves uOrder and vOrder must be greater or equal to 2. An implementation may limit uOrder and vOrder to a certain number. The most common orders are 3 (quadratic polynomial) and 4 (cubic polynomial), which are sufficient to achieve the desired curvature in most cases.
  The number of control points must be at least equal to the order of the curve. The order defines the number of adjacent control points that influence a given control point.

- **controlPoint** defines a set of control points of dimension uDimension * vDimension. This set of points defines a mesh similar to the grid of an ElevationGrid whereas the points do not have a uniform spacing. Depending on the weight-values and the order this hull is approximated by the resulting surface. uDimension points define a polyline in u-direction followed by further u-polylines with the v-parameter in ascending order. The number of control points must be equal or greater than the order. A closed B-Spline surface can be specified by repeating the limiting control points.

The control vertex corresponding to the control point P[i, j] on the control grid is :

```
P[i,j].x = controlPoints[i + (j * uDimension)].x
P[i,j].y = controlPoints[i + (j * uDimension)].y
P[i,j].z = controlPoints[i + (j * uDimension)].z
P[i,j].w = weight[ i + (j * uDimension)]
```
where 0 <= i < uDimension and 0 <= j < vDimension.

- A **weight** value that must be greater than zero is assigned to each controlPoint. The ordering of the values is equivalent to the ordering of the control point values. If the weight of a control point increased above 1 the point is closer approximated by the surface. The number of values must be identical to the number of control points. If the length of the weight vector is 0, the default weight 1.0 is assumed for each control point.

- **uKnots** and **vKnots** define the knot vector. The number of knots must be equal to the number of control points plus the order of the curve. The order must be non-decreasing. By setting successive knot values equal the degree of continuity is decreased, which implies that the surface gets edges. If k is the order of the curve, k consecutive knots at the end or the beginning of the vector let converge the curve to the last or the first control point respectively. Within the knot vector there may be not more than k-1 consecutive knots of equal value. If the length of a knot vector is 0, a default uniform knot vector is computed.

- **uTessellation** and **vTessellation** are ignored by Cortona (used for the compatibility with blaxxun Contact).

- **texCoord** could provide additional information on how to generate texture coordinates.
  By default, texture coordinates in the unit square are generated automatically from the parametric subdivision. The texCoord field specifies per-vertex texture coordinates for the NurbsSurface node. If texCoord is not NULL, it shall specify a TextureCoordinate node containing (uDimension)x(vDimension) texture coordinates; one for each control point, ordered according to a set of control points. The texture coordinates for each point of the NURBS surface are calculated the same way as ordinary coordinates, but the array from TextureCoordinate is used instead of controlPoint.

- **ccw** and **solid** are defined like in other VRML Geometry nodes. solid TRUE enables two-sided lighting, the surface is visible from both sides, and normals are flipped toward the viewer, prior to shading.

- The **distance** and **quality** fields enable to control the quality of the NURBS surface breake down (triangulation) depending on the distance from the camera to the center of the object bounding box. The distance field specifies a set of distances to object. Each distance[i] value is correspond to the quality[i+1] value. The quality[0] value specifies the triangulation quality at distance=0. Thus, if the greatest index in the distance field is N, there shall be N+1 qualities in the quality field.
  Let N denotes the greatest index in the distance field. The following equations define the current quality of the NURBS surface breake down:

  ```
  D <= distance[0]:

  Q = quality[0] + (quality[1] - quality[0]) * D / distance[0]

  distance[i] < D <= distance[i+1]:

  Q = quality[i+1] + (quality[i+2] - quality[i+1]) * (D -
  distance[i]) / (distance[i+1] - distance[i])

  distance[N-1] < D:

  Q = quality[N],
  ```

  where:
  **D** - the distance from the camera to the center of the object bounding box
  **Q** - the current quality of the spline surface brake down, ranging from 0 for the worst quality to 1 for best surface.

## CortonaExtrusion

CortonaExtrusion is an extension of the standard Extrusion node. It allows for preventing the twist of extrusion models that have complicated spines. CortonaExtrusion has preventTwist field, which value determines how the Z-axis of the SCP is computed.

If preventTwist is FALSE, then CortonaExtrusion is identical to the standard Extrusion. The orientation of each cross-section is calculated from the local curvature of the spine. In some cases this algorithm can cause undesirable twists and distortions of the surface.

The Z-axis for points other than the first or last is determined as follows:

```
z = (spine[i+1] - spine[i]) * (spine[i-1] - spine[i])
```

If preventTwist is TRUE, then the orientation of each cross-section (except the first one) is approximately parallel to the orientation of the previous cross-section. This algorithm can help avoid undesirable twists and distortions.
The Z-axis for points other than the first or last is determined as follows:

```
z = x[i-1] * y[i]
```

```
CortonaExtrusion {
   eventIn MFVec2f      set_crossSection
   eventIn MFRotation   set_orientation
   eventIn MFVec2f      set_scale
   eventIn MFVec3f      set_spine
   field        SFBool      preventTwist FALSE
   field        SFBool      beginCap     TRUE
   field        SFBool      ccw          TRUE
   field        SFBool      convex       TRUE
   field        SFFloat     creaseAngle  0
   field        MFVec2f     crossSection [1 1 1 -1 -1 -1  -1  1  1
1]
   field        SFBool      endCap       TRUE
   field   MFRotation   orientation   [0 0 1 0]
   field   MFVec2f      scale         [1 1]
   field   SFBool       solid        TRUE
   field   MFVec3f      spine         [0 0 0 0 1 0]
}
```

## ClippingPlane and ClippingPlaneCanceller

```
EXTERNPROTO ClippingPlane [
   exposedField     SFBool       enabled
   exposedField     SFVec3f      normal
   exposedField     SFFloat      distance
   exposedField     SFNode       appearance
   exposedField     SFBool       cap
]
[
   "urn:inet:parallelgraphics.com:cortona:ClippingPlane"
"http://download.cortona3d.com/public/extensions/extensions.wrl#
ClippingPlane"
]


EXTERNPROTO ClippingPlaneCanceller [

]
```

```
[
  "urn:inet:parallelgraphics.com:cortona:ClippingPlaneCanceller"
"http://download.cortona3d.com/public/extensions/extensions.wrl#
ClippingPlaneCanceller"
]
```

The ClippingPlane node specifies the cross-section plane, which is applied to all geometry nodes on the same level of hierarchy (siblings) and below. ClippingPlane has no effect on group nodes containing the ClippingPlaneCanceller node, which can be used to exclude some of the geometry nodes from the cut.

The *enabled* field turns the section plane on or off.

The *normal* field specifies the normal vector to the plane.

The *distance* field provides distance to the plane from the point (0,0,0).

The *appearance* field specifies the appearance node that is applied to the cross section (if *cap* is set to true).

The *cap* field allows to generate a face on the cross section.

## Layers and Rendering

### Layers and 2D Nodes

#### *Layers*

Layers are transparent rectangular areas on the screen in which VRML scenes are rendered. These areas always face the viewer.

#### *Layer2D*

```
EXTERNPROTO Layer2D [
   eventIn    MFNode addChildren
   eventIn    MFNode removeChildren
   exposedField     MFNode children     NULL
   exposedField     SFVec2f      size          -1, -1
   exposedField     SFNode background   NULL
]
[
   "urn:inet:parallelgraphics.com:cortona:Layer2D"
"http://www.cortona3d.com/source/extensions.wrl#Layer2D"
]
```

The Layer2D node represents an area where 2D scene is rendered.

Its coordinate system's origin is positioned in the center of the rendering area, the x-axis is positive to the right and y-axis in positive upwards.

The width of the rendering area represents -1.0 to +1.0 on the x-axis. The extent of the y-axis in the positive and negative directions is determined by the aspect ratio of the rendering area so that the unit of distance is equal in both directions.

The *children* field may contain any 2D nodes.
The *addChildren* and *removeChildren* fields are lists of 2D nodes to add and, respectively, remove from the layer.

The *size* parameter specifies width and height of layer rectangle in local coordinate system.

The *background* field specifies the Background of the current layer. It may only contain Background2D node.

### Layer3D

```
EXTERNPROTO Layer3D [
   eventIn   MFNode addChildren
   eventIn   MFNode removeChildren
   exposedField    MFNode children     NULL
   exposedField    SFVec2f      size          -1, -1
   exposedField    SFNode background   NULL
   exposedField    SFNode fog          NULL
   exposedField    SFNode navigationInfo     NULL
   exposedField    SFNode viewpoint    NULL
]
[
   "urn:inet:parallelgraphics.com:cortona:Layer3D"
"http://www.cortona3d.com/source/extensions.wrl#Layer3D"
]
```

The Layer3D node represents an area where 3D scene is rendered. Its coordinate system is the same as used in VRML scene.

The *children* field may contain any 3D nodes.
The *addChildren* and *removeChildren* fields are lists of 3D nodes to add and, respectively, remove from the layer.

The *size* parameter specifies width and height of layer rectangle in local coordinate system.

The *background* field specifies the Background of the current layer. It may only contain Background node.

The *fog* field specifies the Fog node.

The *navigationInfo* field specifies the NavigationInfo node.

The *viewpoint* field specifies the Viewpoint node.

## 2D Nodes

2D geometry nodes specify the planar type of geometry nodes. All 2D geometry nodes are used in the two-dimensional coordinate system. The origin and direction of x- and y-axes in the 2D coordinate system coincides with the origin and direction of x- and y-axes in the 3D coordinate system correspondingly. Z-component is set to null (z=0). As 2D geometry nodes come from geometry component they are defined in the geometry field of the Shape node. As all geometry nodes, 2D geometry nodes are affected by the Appearance node, which describes by the appearance properties (material and texture) that is applied to the geometry. Only emissivecolor and transparency of the Material properties are applied to 2D geometry, other properties have no effect.

2D geometry is mainly implemented (designed) for use in Layer2D nodes.

## Circle

```
EXTERNPROTO Circle [
   field SFFloat  radius  #1 (0,inf)
]
[
   "urn:inet:parallelgraphics.com:cortona:Circle"
"http://www.cortona3d.com/source/extensions.wrl#Circle"
]
```

The Circle node specifies a circle centered at (0,0) in the local 2D coordinate system. The *radius* field specifies the radius of the Circle. The value of *radius* should be greater than zero.

## *Rectangle*

```
EXTERNPROTO Rectangle [
   field SFVec2f  size  #2 2 (0,inf)
]
[
   "urn:inet:parallelgraphics.com:cortona:Rectangle"
"http://www.cortona3d.com/source/extensions.wrl#Rectangle"
]
```

The Rectangle node specifies a rectangle centered at (0, 0) in the current local 2D coordinate system and aligned with the local coordinate axes. The *size* field specifies the values of the rectangle's sides. Each component value should be greater than zero.

## *IndexedLineSet2D*

```
EXTERNPROTO IndexedLineSet2D [
   eventIn       MFInt32  set_colorIndex
   eventIn       MFInt32  set_coordIndex
   exposedField  SFNode   color           #NULL
   exposedField  SFNode   coord           #NULL
   field         MFInt32  colorIndex      #[]
   field         SFBool   colorPerVertex  #TRUE
   field         MFInt32  coordIndex      #[]
]
[
   "urn:inet:parallelgraphics.com:cortona:IndexedLineSet2D"
"http://www.cortona3d.com/source/extensions.wrl#IndexedLineSet2D
"
]
```

The IndexedLineSet2D represents a 2D shape consisting of 2D lines. The *coord* field contains the Coordinate2D node that specifies coordinates of the vertices, from which lines are formed. IndexedFaceSet2D is a 2D equivalent of the IndexedLineSet node.

## *IndexedFaceSet2D*

```
EXTERNPROTO IndexedFaceSet2D [
   eventIn       MFInt32  set_colorIndex
   eventIn       MFInt32  set_coordIndex
   eventIn       MFInt32  set_texCoordIndex
   exposedField  SFNode   color           #NULL
   exposedField  SFNode   coord           #NULL
   exposedField  SFNode   texCoord        #NULL
   field         MFInt32  colorIndex      #[]
   field         SFBool   colorPerVertex  #TRUE
   field         SFBool   convex          #TRUE
   field         MFInt32  coordIndex      #[]
   field         MFInt32  texCoordIndex   #[]
]
```

```
[
   "urn:inet:parallelgraphics.com:cortona:IndexedFaceSet2D"
"http://www.cortona3d.com/source/extensions.wrl#IndexedFaceSet2D
"
]
```

The IndexedFaceSet2D represents a 2D shape consisting of 2D faces. The *coord* field contains the Coordinate2D node that specifies coordinates of the vertices from which faces are formed. IndexedFaceSet2D is a 2D equivalent of the IndexedFaceSet node.

### *PointSet2D*

```
EXTERNPROTO PointSet2D [
   exposedField  SFNode color   #NULL
   exposedField  SFNode coord   #NULL
]
[
   "urn:inet:parallelgraphics.com:cortona:PointSet2D"
"http://www.cortona3d.com/source/extensions.wrl#PointSet2D"
]
```

The PointSet2D node specifies a set of 2D points. The *coord* field contains the Coordinate2D node. The PointSet2D node is a 2D equivalent of the PointSet node.

### *Coordinate2D*

```
EXTERNPROTO Coordinate2D [
   exposedField  MFVec2f  point   #[]
]
[
   "urn:inet:parallelgraphics.com:cortona:Coordinate2D"
"http://www.cortona3d.com/source/extensions.wrl#Coordinate2D"
]
```

The Coordinate2D node specifies a set of 2D coordinates, which is used in the *coord* field of PointSet2D, IndexedLineSet2D and IndexedFaceSet2D nodes.

### *CoordinateInterpolator2D*

```
EXTERNPROTO CoordinateInterpolator2D [
   eventIn       SFFloat  set_fraction
   exposedField  MFFloat  key       #[]
   exposedField  MFVec2f  keyValue  #[]
   eventOut      MFVec2f  value_changed
]
[
"urn:inet:parallelgraphics.com:cortona:CoordinateInterpolator2D"
"http://www.cortona3d.com/source/extensions.wrl#CoordinateInterp
olator2D"
]
```

The CoordinateInterpolator2D node is the 2D equivalent of the CoordinateInterpolator node.

### *Transform2D*

```
EXTERNPROTO Transform2D [
   eventIn      MFNode   addChildren
   eventIn      MFNode   removeChildren
   exposedField SFVec2f  center           #0,0
   exposedField MFNode   children         #[]
   exposedField SFFloat  rotationAngle    #0.0
   exposedField SFVec2f  scale            #1,1
   exposedField SFFloat  scaleOrientation #0.0
   exposedField SFVec2f  translation      #0,0
]
[
   "urn:inet:parallelgraphics.com:cortona:Transform2D"
"http://www.cortona3d.com/source/extensions.wrl#Transform2D"
]
```

The Transform2D node is a 2D equivalent of the Transform node. It is a grouping node that allows translation, rotation and scaling of its 2D children. The *translation* field specifies translation of the children objects.
The *rotationAngle* field specifies rotation of the children objects. The centre of rotation is the point specified in the *center* field. The *scale* field specifies scaling of the children nodes. The *scaleOrientation* specifies a rotation of the coordinate system before the scale (to specify scales in arbitrary orientations). The *scaleOrientation* field applies only to the scale operation.

### *Transform2Dex*

The Transform2DEx node allows for positioning layers on the screen and specifying their size in pixels.

Note: The Transform2DEx node should not be used inside Transform and Layer nodes.

```
EXTERNPROTO Transform2DEx [
   eventIn      MFNode   addChildren
   eventIn      MFNode   removeChildren
   exposedField SFVec2f  center           #0,0
   exposedField MFNode   children         #[]
   exposedField SFFloat  rotationAngle    #0.0
   exposedField SFVec2f  scale            #1,1
   exposedField SFFloat  scaleOrientation #0.0
   exposedField SFVec2f  translation      #0,0
   exposedField SFVec2f  origin        #0,0
   exposedField SFVec2f pixelTranslation     #0,0
   exposedField SFVec2f pixelScale    #0,0
]
[
   "urn:inet:parallelgraphics.com:cortona:Transform2DEx"
"http://www.cortona3d.com/source/extensions.wrl#Transform2DEx"
]
```

The Transform2DEx node is a equivalent of the Transform2D node. It is a grouping node that allows translation, rotation and scaling of its 2D children.
The *translation* field specifies translation of the children objects.
The *rotationAngle* field specifies rotation of the children objects.
The centre of rotation is the point specified in the *center* field.
The *scale* field specifies scaling of the children nodes.
The *scaleOrientation* specifies a rotation of the coordinate system before the scale (to specify scales in arbitrary orientations). The *scaleOrientation* field applies only to the scale operation.
The *origin* field specifies the translation of coordinate system.
The *pixelTranslation* field specifies translation of the children objects in pixels.
The *pixelScale* field specifies scaling of the children objects in pixels.

### Background2D

```
EXTERNPROTO Background2D [
   eventIn        SFBool   set_bind
   exposedField   SFColor  backColor #0  0  0
   exposedField   MFString url       #[]
   eventOut       SFBool   isBound
]
[
"urn:inet:parallelgraphics.com:cortona:Background2D"
"http://www.cortona3d.com/source/extensions.wrl#Background2D"
]
```

Background2D node is a 2D equivalent of the Background node.
Background2D is used only in 2D context, such as Layer2D node.

*backColor* field specifies the color of the of the background. *url* field specifies an image which is applied to the 2D background.

As there is no background stack in the layer
nodes, *set_bind* and *isBound* fields are ignored.

## Panel and HTMLText

### Panel

```
EXTERNPROTO Panel [
   exposedField   SFNode         source          #NULL
   exposedField  SFString      left  #"0"
   exposedField  SFString      top  #"0"
   exposedField  SFString      right  #""
   exposedField  SFString      bottom  #""
   exposedField  SFString      width  #""
   exposedField  SFString      height  #""
   exposedField  SFString      offsetLeft  #""
   exposedField  SFString      offsetTop  #""
   exposedField  SFBool        sticky  #FALSE
   exposedField  SFBool        enabled  #FALSE
   exposedField  SFFloat       backgroundTransparency  #1
   exposedField  SFColor       backgroundColor  #1 1 1
   exposedField  SFInt32       borderSize  #0
   exposedField  SFColor       borderColor  #1 1 1
   eventOut        MFInt32       contentSize
   eventOut        SFTime        touchTime
   eventOut        SFVec2f       touchPoint
   eventOut        SFString      hotspot
   eventOut        SFTime        hotspotTime
]
[
   "urn:inet:parallelgraphics.com:cortona:Panel"
"http://www.cortona3d.com/source/extensions.wrl#Panel"
]
```

The Panel node represents a rectangular area where HTML text can be rendered. This area always faces the viewer.
The *source* field contains an HTMLText node or NULL.
The *left*, *to*, *right* and *bottom* fields specify panel coordinates. Each coordinate can be specified in pixels, in percents of width (for the left or right fields) or height (for the top or bottom) of 3D window size or can be omitted.
The *width* and *height* fields specify the width and height of the panel correspondingly. Width and height can be specified in pixels, in percents of 3D window size or can be omitted.
The *offsetLeft* and *offsetTop* fields specify the offset of the panel's content. offsetLeft, offsetTop can be specified in pixels, in percents of panel's size or can be omitted.
The *sticky* field specifies if the parent coordinate system is used or not. TRUE value means that left, right, top and bottom are ignored and upper left corner of the panel is positioned at the origin of the parent transform.
The *enabled* field specifies if mouse events are processed or not. FALSE value means that no mouse events are processed; TRUE value means that all mouse events are processed.
The *backgroundTransparency* field specifies the transparency of the panel's background.
The *backgroundColor* field specifies the color of the panel's background.
The *borderSize* field specifies the size of the panel's border.
The *borderColor* field specifies the color of the panel's border.
The *contentSize* event is generated if the size of the panel is changed. ContentSize event value contains the size of the panel in pixels.

The *touchTime* and *TouchPoint* events. If enabled field is set to TRUE, TouchTime and TouchPoint events are generated when user clicks on the panel's area.
The *hotspot* and *hotspotTime* events. If enabled field is set to TRUE, hotspot and hotspotTime events are generated when user clicks on the <a href=""></a> tag. Hotspot contains the href of the corresponding <a> tag value.

### *HTMLText*

```
EXTERNPROTO HTMLText [
   exposedField  SFString     body  #""
   exposedField  MFInt32      padding  #[]
   exposedField  SFBool       shadow  #FALSE
   exposedField  SFFloat      shadowTransparency  #0
   exposedField  SFInt32      shadowSize  #8
   exposedField  MFInt32      shadowOffset  #[]
   exposedField  SFColor      shadowColor  #0 0 0
]
[
   "urn:inet:parallelgraphics.com:cortona:HTMLText"
"http://www.cortona3d.com/source/extensions.wrl#HTMLText"
]
```

The HTMLText node represents an HTML text, which can be used in the Panel node. The *Body* field contains a string of HTML code. Only p, a, font, b (strong), i (em), u, br, center tags with the face, size, color attributes are supported.
The *padding* field specifies padding in pixels in a form of [top [right [bottom [left]]]].
The *Shadow* field specifies if shadow is used or not. FALSE value means that no shadow is used, TRUE value means that shadow is used.
The *ShadowTransparency* field specifies the transparency of the shadow.
The *ShadowSize* field specifies the size of the shadow.
The *ShadowOffset* field specifies the shadow's offset.
The *ShadowColor* field specifies the shadow's color.

## OrderedGroup

```
EXTERNPROTO OrderedGroup [
   eventIn       MFNode   addChildren
   eventIn       MFNode   removeChildren
   exposedField  MFNode   children  #[]
   exposedField  MFFloat  order     #[]
]
[
   "urn:inet:parallelgraphics.com:cortona:OrderedGroup"
"http://www.cortona3d.com/source/extensions.wrl#OrderedGroup"
]
```

OrderedGroup node is a grouping node which allows to set the order of the rendering of the coplanar or close shapes.

As in all grouping nodes, *children* field specifies a list of children nodes of the OrderedGroup node, *addChildren* and *removeChildren* fields specify the list of objects that shall be added or, respectively, removed from the OrderedGroup node.

*order* field is an array of floating point numbers. Each value of the *order* field corresponds to one child from the children field. The child that has the lowest order value is rendered first. Other children are rendered in increasing order. The last rendered child is a child with the highest order value. If the *order* field is empty, all the children of the OrderedGroup are rendered in the order, that is specified in the children field, from the first to the last.

## ZGroup

The **ZGroup** node enables/disables writing children geometry in Z-buffer and checking Z-buffer during its output.

```
EXTERNPROTO    ZGroup [
     eventIn        MFNode    addChildren
     eventIn        MFNode    removeChildren
     exposedField   MFNode    children []
     field          SFVec3f   bboxCenter 0 0 0     # (-inf inf)
     field          SFVec3f   bboxSize   -1 -1 -1   # (0, inf) or -
1,-1,-1
     exposedField   SFBool    write TRUE
     exposedField   SFBool    check TRUE
     exposedField   SFBool    writePixelBuffer TRUE
]
[
     "urn:inet:parallelgraphics.com:cortona:ZGroup"
"http://www.cortona3d.com/source/extensions.wrl#ZGroup"
]
```

The fields of the ZGroup node, with the exception of the check, and write fields, are analogous to the corresponding fields of the Group VRML node.

Fields and events:

**Check** field specifies whether Z-buffer is checked during rendering of geometry specified in the children field.

**Write** field specifies whether the children geometry should be written to Z-buffer.

**WritePixelBuffer** field specifies whether the children geometry should be written to pixel-buffer. The default value of writePixelBuffer is TRUE.

## Text

### Three-Dimensional Text

#### *The Text3D node*

Incorporate 3D text into your VRML world and format it with any True Type font installed in your Windows system.

##### Node description

```
Text3D {

  exposedField MFString string []

  exposedField SFNode fontStyle NULL

  exposedField MFFloat length []   # [0, inf)

  exposedField SFFloat maxExtent 0 # [0, inf)

  exposedField SFFloat depth 0.1   # [0, inf)

  exposedField SFFloat creaseAngle 0    # [0, inf)

  exposedField SFBool solid TRUE

}
```

You can incorporate 3D text into your VRML world and format it with any True Type font installed in your Windows system. The **Text3D** node specifies a 3D text string object that is positioned with its middle vertical plane in the Z=0 plane of the local coordinate system, based on values defined in the **fontStyle** field. The **Text3D** nodes may contain multiple text strings using the UTF-8 encoding as specified by ISO 10646-1:1993. The text strings are stored in the order in which the text mode characters are to be produced as defined by the parameters in the **FontStyle** node. The fields of the **Text3D** node, with the exception of **depth**, **creaseAngle** and **solid**, are analogous to the ones for the **Text** node.

The **depth** field contains a SFFloat value that specifies the thickness of each text string in the local coordinate system.

The **creaseAngle** field affects how default normals are generated. If the angle between the geometric normals of two adjacent faces is less than or equal to the specified value for the crease angle parameter, the edge between the two adjacent faces is smooth-shaded. Otherwise, the appearance of a rendered surface is calculated so that a lighting discontinuity is produced across the edge.

The **solid** field determines whether one or both sides of each polygon should be displayed. If solid is FALSE, each polygon of 3D text will be visible regardless of the viewing direction. If solid is TRUE, this results in one-sided polygon lighting.

Descriptions of the string, fontStyle, length, and maxExtent fields are provided in the VRML97 specification (see http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#Text).

### Example

```
#VRML V2.0 utf8
NavigationInfo {
  type "EXAMINE"
}
  DEF MainTransform Transform {
   children Shape {
    geometry Text3D {
    string ["@"]    fontStyle FontStyle {
    justify ["MIDDLE", "MIDDLE"]
    family "Times"
    style "BOLD"
    size 4
    }
    depth 0.5
   }
   appearance Appearance {
    material Material {
     diffuseColor .28 .42 .6
     specularColor .32 .4 .4
     ambientIntensity .05
     shininess .54
     emissiveColor .14 .22 .31
    }
   }
  }
 }

  DEF MainInterpolator OrientationInterpolator {
  key [ 0 0.5 1 ]
  keyValue [0 1 0 0 0 1 0 3.14 0 1 0 6.28]
 }
  DEF MainTimer TimeSensor {
  loop TRUE
  cycleInterval 5
 }
 ROUTE MainTimer.fraction_changed TO MainInterpolator.set_fraction
 ROUTE MainInterpolator.value_changed TO MainTransform.rotat
```

## FontStyle

### *Enhanced support for the FontStyle node*

As well as specifying the font family rendering techniques, Cortona 3.1 allows the developer to select a specific font based on the font name. You can incorporate a flat text string object into your VRML world and format it with any True Type font installed in your Windows system. For example, family "Times New Roman".

```
...
   geometry Text3D {
    string ["ÃŸ"]
    fontStyle FontStyle {
    justify ["MIDDLE", "MIDDLE"]
    family [ "Verdana", "Arial", "Helvetica" ]
    language "238"
    style "BOLD"
    size 4
    }
    creaseAngle 1.5
    depth 0.5
   }
...
```

The language field provides a proper language attribute of the text string. The following table represents codes for the representation of names of languages that Cortona supports. Both two-letter symbol and character set specify the language to use:

| Two-letter symbol | MS Charset | MS Charset Name | Language |
|---|---|---|---|
| "ar" | "178" | ARABIC_CHARSET | Arabic |
| "el" | "161" | GREEK_CHARSET | Greek |
| "he" | "177" | HEBREW_CHARSET | Hebrew |
| "ja" | "128" | SHIFTJIS_CHARSET | Japanese |
| "ko" | "129" | HANGUL_CHARSET | Korean |
| "zh" | "136" | CHINESEBIG5_CHARSET | Chinese |
| "ru" | "204" | RUSSIAN_CHARSET | Russian |
| "tr" | "162" | TURKISH_CHARSET | Turkish |
| no | "238" | EE_CHARSET | Eastern Europe |
| no | "2" | SIMBOL_CHARSET | Symbol fonts |

You can also use any truly numerical value for the Microsoft character set even if it is not listed in the table.

For the multilingual support the string field in the Text node should contain text strings that are specified in UTF-8.

## Mouse and keyboard input

### Drag & Drop Handling

#### *DropSensor*

The **DropSensor** node generates events based on input from a pointing device. Retrieves an object's uniform resource locator (URL) of an object (resource) dragged to the 3D window.

To texture an object, drag and drop the texture or links (in Netscape Navigator) onto the box in the 3D window. You can also drag a link to any of image file from your local drive or Internet.

#### *Node description*

```
DropSensor {
   exposedField     SFBool     enabled TRUE
   eventOut    SFVec3f   hitPoint
   eventOut    SFVec3f   hitNormal
   eventOut    SFVec2f   hitTexCoord
   eventOut    SFTime    dropTime
   eventOut    MFNode    nodeChain
   eventOut    MFString  url
}
```

**enabled** field indicates whether the sensor is currently paying attention to pointing device input.

**hitPoint** - the location on the surface of the underlying geometry at which the primary button of the pointing device was released.

**hitNormal** - the normal at the point given by hitPoint.

**hitTexCoord** - the texture coordinate at the point given by hitPoint.
**dropTime** - the time at which the primary button of the pointing device was released.

**nodeChain** returns the nodes names from the top-level to the geometry at which the primary button of the pointing device was released.

**url** returns the URL for the object (resource) currently dragged to the 3D window.

### *Example*

```
#VRML V2.0 utf8
NavigationInfo {
  type "EXAMINE"
}
Transform {
  rotation 1 1 1 1
  children [
   DEF DS DropSensor {}
   Shape {
    geometry Box {}
    appearance Appearance {
     texture DEF IT ImageTexture {}
     material Material {
      shininess 1
     }
    }
   }
  ]
 }

ROUTE DS.url TO IT.url
```

## Keyboard Input

### *KbdSensor*

The **KbdSensor** node generates events based on input from a keyboard.

Click anywhere in the 3D window and press any alphanumeric keys.

### *Node description*

```
KbdSensor {
  exposedField      SFBool enabled TRUE
  exposedField      SFBool isActive FALSE
  eventOut          SFInt32      keyDown
  eventOut          SFInt32      keyUp
}
```

**enabled** indicates whether the sensor is currently paying attention to a keyboard input. If enabled receives TRUE and isActive is TRUE, the sensor reacts on input from a keyboard. In this case, the user cannot navigate in the 3D window using keyboard commands.

**isActive** allows to control the sensor. If isActive receives a TRUE event, the sensor processes all keyboard input. Otherwise, the input is treated with a browser.

**keyDown** and **keyUp** events generate 32-bit value containing the character code of the key that was pressed or released. The primary two bytes specify the virtual-key code of the nonsystem key, and the secondary - key-state flags

## *Example*

```
#VRML V2.0 utf8
NavigationInfo {
  type "EXAMINE"
}
Background {
  skyColor [1 1 1]
}

DEF KS KbdSensor {
  isActive TRUE
}
Shape {
  geometry DEF TXT Text {
   string ["?"]
   fontStyle FontStyle   {
   justify ["MIDDLE", "MIDDLE"]
   family "TYPEWRITER"
   style "BOLD"   size 4
   }
  }
  appearance Appearance {
   material Material {
    diffuseColor 0.02 0.38 0.61
   }
  }
}

DEF SCR Script {
  directOutput TRUE
  eventIn SFInt32 go
  field SFNode TXT USE TXT
  field MFString string [""]
  url ["javascript:
   function go(val,ts){
    string[0]=
     String.fromCharCode(val.toString());
    TXT.string=string;
   }
   "]
```

```
 }

ROUTE KS.keyDown TO SCR.go
```

# Movies

## Animated GIF Files as Source for MovieTexture

Use the MovieTexture node to place animated GIF files in your VRML scene.

```
#VRML V2.0 utf8
     DEF ROTY Transform {
      children DEF ROTX Transform {
       children DEF ROTZ Transform    {
         children     Transform {
            rotation 0 0 1 0 children [
             Shape {appearance Appearance {
              texture MovieTexture {
               loop TRUE
          url "banner.gif"
        }
      }
             geometry Box {size 8.8 3.1 0.1}}

      DEF TS1 TouchSensor {}]
   }
  }}}

DEF ROTYInterpolator OrientationInterpolator {
  key [ 0 0.5 1 ]
  keyValue [0 1 0 0 0 1 0 -3.14 0 1 0 -6.28]
}

DEF ROTZInterpolator OrientationInterpolator {
  key [ 0 0.5 1 ]
  keyValue [0 0 1 0, 0 0 1 -0.8, 0 0 1 0]
}

  DEF ROTXInterpolator OrientationInterpolator {
  key [ 0 0.5 1 ]
  keyValue [1 0 0 0, 1 0 0 0.8, 1 0 0 0]
}

  DEF TIMER TimeSensor { enabled TRUE
    loop FALSE cycleInterval 10}

ROUTE TS1.touchTime TO TIMER.startTime
ROUTE TIMER.fraction_changed TO ROTYInterpolator.set_fraction
ROUTE TIMER.fraction_changed TO ROTZInterpolator.set_fraction
ROUTE TIMER.fraction_changed TO ROTXInterpolator.set_fraction
ROUTE ROTYInterpolator.value_changed TO ROTY.rotation
ROUTE ROTZInterpolator.value_changed TO ROTZ.rotation
ROUTE ROTXInterpolator.value_changed TO ROTX.rotation
```

# Behaviour

## Object-To-Object Collision Detection Interface

Object-to-object collision detection in a three-dimensional scene is a procedure of determining whether a given shape, if it were to undergo some transformation (for example, to be moved, rotated, or scaled), would encounter an obstacle in the form of another shape. In this document we describe the ParallelGraphics' ECMAScript interface with the proprietary implementation of the object-to-object collision detection extension to VRML.

The interface is built around two native ECMAScript objects, **Collidee** and **Collision**. The former acts as a proxy for the shape that is transformed, bearing the parameters of the transformation matrix and other relevant data, and the latter describes the point of the shape in question that came into contact with another shape, in the case of a collision. Let us examine each object in turn.

### *Object Collidee*

**Properties:**

| | |
|---|---|
| SFNode/MFNode | **body** |
| SFVec3f | **position** |
| SFRotation | **orientation** |
| SFVec3f | **scale** |
| SFVec3f | **size** |
| SFVec3f | **offset** |
| Collision | **collision** (read-only) |
| Collidee | **scenery** |
| SFNode/MFNode | **ignore** |

**Methods:**
Boolean **moveTo**(SFVec3f position, SFRotation orientation, SFVec3f scale) (all parameters are optional)

- **Body**, **position**, **orientation**, **scale**, **size** and **offset**
  The **body** property contains a reference to the shape or a list of shapes that are subjected to collision detection and the **position**, **orientation**, and **scale** properties store references to the parameters of the matrix used for transforming the coordinates of elements of that shape. When the **body** property is set to **null**, the **size** and **offset** properties are used to construct an imaginary

shape of parallelepiped with edges parallel to basis vectors, and the center displaced from the origin of the local coordinate frame.

- **Collision**
  The **collision** property references an object describing the contact point of the shape during the last collision.

- **Scenery**
  The **scenery** property contains a reference to another **Collidee** object against which the collision detection is performed; if this property is set to **null**, then every shape in the scene is used.

- **Ignore**
  The **ignore** property is a reference to a shape or a list of shapes that should not be processed when detecting collision with the current shape, which itself is considered to be ignored for this purpose.

- **moveTo**
  The **moveTo** method does the job of collision detection when transforming the shape that a given **Collidee** object represents. It builds two transformation matrices for the shape, one from the parameters in the **Collidee** object, and the other from the arguments received, and checks that no collision occurs at both the initial and final positions, or at any position interpolated between these two. In the case of no collision the method copies the values of arguments to the corresponding properties of the **Collidee** object and returns **true**. Otherwise, a transformation matrix corresponding to the position of the shape when it first comes into contact with an obstruction is computed, the properties of the Collision object are updated with the appropriate values, and the method returns **false**. Also, in this case, the properties of the **Collision** object are set to the values describing the contact point of the shape. It should be noted that for optimization purposes the values of
  the **position**, **orientation** and **scale** properties are never changed by the **moveTo** method; instead, the values of the objects that these properties point to are updated. In practice this makes it possible for the **translation**, **rotation** and **scale** fields of the corresponding Transform node to be automatically updated as a side-effect of a **moveTo** call.

### *Object Collision*

**Properties:**

| | | |
|---|---|---|
| SFVec3f | **Point** | (read-only) |
| SFVec3f | **Normal** | (read-only) |
| Number | **faceIndex** | (read-only) |
| MFNode | **Path** | (read-only) |

- **point** and **normal**The **point** property contains the coordinates, relative to the local coordinate frame of the shape, of the point that contacts the obstruction, and the **normal** property gives the coordinates of the normal vector at that point.

- **faceIndex**The **faceIndex** property indicates which face of an IndexedFaceSet contains the specified point; this value is not defined for other geometry nodes.

- **Path**The **path** property is a list of nodes forming a chain in the hierarchy of nodes starting from the one specified in Collidee's body property, and ending with the node that contains the face that collided.

The described interface is, from the programmer's point of view, a filter for the transformations that are sent through it to the shape that it governs. This fact makes the use of collision detection more or less transparent to the author of VRML scenes and even allows for the augmentation of the Interpolator-based animations with collision-detection techniques.

## NavigationInfo

According to the VRML97 Specification, the first three values (zero, first, and second elements) of the *avatarSize* MFFloat-typed field of the NavigationInfo node define the avatar's physical dimensions in the scene for the purpose of collision detection and terrain following. In Cortona VRML Client it is possible to use further values contained by the *avatarSize* field to customize navigation in the EXAMINE navigation mode and specify an advanced rendering parameter:

- The second triple of values in the field (third, fourth and fifth elements) sets an arbitrary position of the center of scene rotation in the EXAMINE navigation mode, its x, y, and z coordinates. (By default, the center of rotation in this mode in Cortona coincides with the center of the bounding box of the scene geometry).

- The seventh value in the field (sixth element) specifies the near visibility limit for the improvement of Z-buffer accuracy. Geometry before the near visibility limit will not be rendered. The use of this value is similar to the *visibilityLimit* field of the NavigationInfo node whose value limits the rendered part of the scene from outside (outer visibility limit). The combined use of the two visibility limits can eliminate Z-buffer problems, which can occur when geometry objects situated very close and very far from the viewer are simultaneously rendered in Cortona.

In the example below, the centre of rotation in the EXAMINE mode is moved to a point with coordinates (3, 3, 0), and the near visibility limit is set to 1 metre.

```
NavigationInfo {
	type ["EXAMINE"]
	avatarSize [0.25, 1.6, 0.75, 3, 3, 0, 1]
}
```

Note: the values of the center of rotation and the near visibility limit can be changed dynamically using scripting capabilities in Cortona.

## Interpolator Nodes in Cortona

The standard VRML interpolator nodes, such as, ColorInterpolator, CoordinateInterpolator, NormalInterpolator, OrientationInterpolator, PositionInterpolator, ScalarInterpolator nodes, and ParallelGraphics Position2Interpolator node (see SFVec2f Interpolator) are designed for linear keyframed animation among the lists of SFColor, SFVec3f, SFRotation, SFFloat and SFVec2f values. These nodes are used as a base for most simple animations in VRML scenes. The implementation of smooth (non-linear) animations in VRML is possible only with the use of the Script VRML node. Apart from being inconvenient for content creators, this results in significant reduction in performance. If smooth animations are created with the use of linear interpolators, a larger number of keyframes and therefore substantially larger size of files result.

Five new ParallelGraphics VRML extension nodes:

- ColorInterpolatorEx,
- OrientationInterpolatorEx,
- PositionInterpolator2Dex,
- PositionInterpolatorEx,
- ScalarInterpolatorEx

were introduced to resolve this problem by extending the capabilities of existing interpolator nodes of the corresponding types. By setting the value of the *type* field of the new interpolator nodes, the content creator can choose the desired type of interpolation - how the values of a parameter should be generated between keyframe values. Specifying non-linear interpolation types allows developers to create smooth and realistic animations with the use of a minimum number of keyframes.

All the new ParallelGraphics interpolator nodes, except for the OrientationInterpolatorEx node, share the following common set of fields and semantics:

```
eventIn        SFFloat    set_fraction
exposedField   MFFloat    key            [...]
exposedField   MF<type>   keyValue       [...]
eventOut       SF<type>   value_changed
exposedField   SFString   type           "LINEAR"
exposedField   MFFloat    params         [0, 0, 0] # (-inf,inf)
```

The *type* field specifies the type of interpolation used. The following values of this field are possible: "CONSTANT", "LINEAR", "COSINE", "CUBIC" and "HERMITE":

| **Value of the *type* field** | **Interpolation** |
|---|---|
| "CONSTANT" | The value remains fixed until the next keyframe. No interpolation is performed. |
| "LINEAR" | The value changes linearly from the previous to the next keyframe value. For complex values, each of the components changes independently of other components. The speed of the value change (acceleration or deceleration) is constant throughout the interval. This is how interpolation is performed by standard VRML interpolator nodes which the corresponding ParallelGraphics interpolator nodes extend. |
| "COSINE" | The value changes according to the cosine law in the interval between the previous and the next keyframe values. The speed of the value change is minimum (zero) both at the beginning and end of the interval while the maximum speed of the value change is achieved in the middle of the interval. |
| "CUBIC" | The value changes between keyframes values according to cubic law. Cubic splines provide quick and smooth interpolation of values. |
| "HERMITE" | The Kochanek-Bartels splines (also known as TCB splines), which are based on Hermite polynoms, provide cubic interpolation of the parameter between keyframes values. The exact type of dependence between the values of a key and the corresponding keyValue parameter (interpolation function) can be customized using three TCB-splines parameters specified by the params field. |

Notes (for "CUBIC" and "HERMITE" interpolation types):

- Linear interpolation is used instead of cubic interpolation if the number of keyframes is less than 4.
- If the value of the first keyframe coincides with the value of the last keyframe, the cubic spline is "closed", i.e. there is no derivative hit of keyValue when interpolating between the last and first keyframes.

Three values in the *params* field of the interpolator nodes with extended capabilities specify the parameters of Kochanek-Bartels splines which customize the interpolation function:

| **Parameter** | **Name** | **Description** |
|---|---|---|
| 0 | Tension | Specifies the bending sharpness of the interpolation function at keyframes (higher tension values correspond to tighter function curves, lower - to looser curves). |
| 1 | Continuity | Specifies the variation in derivative of the interpolation function from the left and right at keyframes (zero - transition between adjacent intervals at keyframes is smooth, non-zero - intersections of intervals are abrupt). |
| 2 | Bias | Controls the amount that the interpolation function bends at each end of the interval between keyframes (Bias<0 - the function bends more at the beginning of the interval, Bias>0 - the function bends more at the end). |

Note: if the values of the Continuity and Bias parameters are equal to 0, the Kochanek-Bartels splines are identical to cardinal splines. If all three parameters are equal to zero (the default value of the params field), the TSB splines are identical to the Catmull-Rom splines.

The OrientationInterpolatorEx node has only one additional field - *type*. This field can take on the following values: "CONSTANT", "LINEAR" (slerp) and "SPLINE" (squad) that specify constant, linear and cubic spline interpolation respectively.

Definitions of the other fields and events in the interpolator nodes are similar to the definitions of the corresponding standard VRML nodes. More information about interpolators in VRML is available at the Interpolators nodes topic in the VRML97 Specification (see http://www.web3d.org/documents/specifications/14772/V2.0/part1/concepts.html#4.6.8).

## *ColorInterpolatorEx Node*

The ColorInterpolatorEx node interpolates among a list of MFColor values using a specified interpolation method. This node extends the standard ColorInterpolator VRML node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#ColorInterpolator.

```
EXTERNPROTO ColorInterpolatorEx [
   eventIn         SFFloat    set_fraction                    # (-
inf,inf)
   exposedField    MFFloat    key              []             # (-
inf,inf)
   exposedField    MFColor    keyValue         []             # [0,1]
   exposedField    MFFloat    params           [0, 0, 0]      # (-
inf,inf)
   exposedField    SFString   type             "LINEAR"
   eventOut        SFColor    value_changed
]
[
   "urn:inet:parallelgraphics.com:cortona:Position2Interpolator"
"http://www.cortona3d.com/source/extensions.wrl#ColorInterpolatorEx"
]
```

The *type* field defines the interpolation method. The possible values of this field are "CONSTANT", "LINEAR", "COSINE", "CUBIC" and "HERMITE". In the case of the Hermite interpolation, the *params* field specifies three interpolation parameters: Tension, Continuity and Bias.

Definitions of the other fields and events of the ColorInterpolatorEx node are similar to the corresponding definitions for the ColorInterpolator VRML node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#ColorInterpolator.

### *OrientationInterpolatorEx Node*

The OrientationInterpolatorEx node interpolates among a list of rotation values using a specified interpolation method. This node extends the standard OrientationInterpolator node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#OrientationInterpolator.

```
EXTERNPROTO OrientationInterpolatorEx [
    eventIn         SFFloat       set_fraction                   # (-
inf,inf)
   exposedField    MFFloat       key              []             # (-
inf,inf)
   exposedField    MFRotation    keyValue         []             # [-
1,1],(-inf,inf)
   exposedField    SFString      type             "LINEAR"
   eventOut        SFRotation    value_changed
]
```

```
[
"urn:inet:parallelgraphics.com:cortona:OrientationInterpolatorEx
"
"http://www.cortona3d.com/source/extensions.wrl#OrientationInter
polatorEx"
]
```

The *type* field defines the interpolation method. The possible values of this field are "CONSTANT", "LINEAR" (slerp) and "SPLINE" (squad):

| **Value of the *type* field** | **Interpolation** |
|---|---|
| "CONSTANT" | The orientation value remains fixed until the next keyframe. No interpolation is performed. |
| "LINEAR" | The value of orientation is interpolated uniformly along a geodesic in the surface of the 3-sphere between the previous and the next keyframe values. This method is often referred to as SLERP (Spherical-Linear intERPolation). That is how the interpolation is made by the OrientationInterpolator node (http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#OrientationInterpolator). |
| "SPLINE" | The value of orientation is interpolated between keyframe values using cubic Hermite polynoms. This method is referred to as SQUAD (Spherical QUADrilateral interpolation). Unlike SLERP, the transition between adjacent intervals at keyframes is smooth. |

Notes (for the "SPLINE" interpolation type):

- Linear interpolation is used instead of cubic interpolation if the number of keyframes is less than 4.
- If the value of the first keyframe coincides with the value of the last keyframe, the cubic spline is "closed".

Definitions of the other fields and events of the OrientationInterpolatorEx node are similar to the corresponding definitions for the standard OrientationInterpolator node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#OrientationInterpolator.

### *PositionInterpolator2DEx Node*

The PositionInterpolator2DEx node interpolates among a list of SFVec2f values using a specified interpolation method. This node extends the ParallelGraphics extension node Position2Interpolator (see SFVec2f Interpolator section above).

```
EXTERNPROTO PositionInterpolator2DEx [
   eventIn          SFFloat   set_fraction                  # (-
inf,inf)
   exposedField     MFFloat   key              []           # (-
inf,inf)
   exposedField     MFVec2f   keyValue         []           # (-
inf,inf)
   exposedField     MFFloat   params           [0, 0, 0]    # (-
inf,inf)
   exposedField     SFString  type             "LINEAR"
   eventOut         SFVec2f   value_changed
]
[
"urn:inet:parallelgraphics.com:cortona:PositionInterpolator2DEx"
"http://www.cortona3d.com/source/extensions.wrl#PositionInterpol
ator2DEx"
]
```

The type field defines the interpolation method. The possible values of this field are "CONSTANT", "LINEAR", "COSINE", "CUBIC" and "HERMITE". In the case of the Hermite interpolation, the params field specifies three interpolation parameters: Tension, Continuity and Bias.

Definitions of the other fields and events of the PositionInterpolator2DEx node are similar to the corresponding definitions for the ParallelGraphics extension node Position2Interpolator (see SFVec2f Interpolator section).

### *PositionInterpolatorEx Node*

The PositionInterpolatorEx node interpolates among a list of 3D vectors using a specified interpolation method. This node extends the standard PositionInterpolator VRML node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#PositionInterpolator.

```
EXTERNPROTO PositionInterpolatorEx [
   eventIn          SFFloat   set_fraction                  # (-
inf,inf)
   exposedField     MFFloat   key              []           # (-
inf,inf)
   exposedField     MFVec3f   keyValue         []           # (-
inf,inf)
   exposedField     MFFloat   params           [0, 0, 0]    # (-
inf,inf)
   exposedField     SFString  type             "LINEAR"
   eventOut         SFVec3f   value_changed
]
[
"urn:inet:parallelgraphics.com:cortona:PositionInterpolatorEx"
"http://www.cortona3d.com/source/extensions.wrl#PositionInterpol
atorEx"
]
```

The type field defines the interpolation method. The possible values of this field are "CONSTANT", "LINEAR", "COSINE", "CUBIC" and "HERMITE". In the case of the Hermite interpolation, the params field specifies three interpolation parameters: Tension, Continuity and Bias.

Definitions of the other fields and events of the ColorInterpolatorEx node are similar to the corresponding definitions for the standard PositionInterpolator VRML node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#PositionInterpolator.

### ScalarInterpolatorEx Node

The ScalarInterpolatorEx node interpolates among a list of SFFloat values using a specified interpolation method. This node extends the standard ScalarInterpolator VRML node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#ScalarInterpolator.

```
EXTERNPROTO ScalarInterpolatorEx [
   eventIn          SFFloat    set_fraction                  # (-
inf,inf)
   exposedField     MFFloat    key              []           # (-
inf,inf)
   exposedField     MFFloat    keyValue         []           # (-
inf,inf)
   exposedField     MFFloat    params           [0, 0, 0]    # (-
inf,inf)
   exposedField     SFString   type             "LINEAR"
   eventOut         SFFloat    value_changed
]
[
   "urn:inet:parallelgraphics.com:cortona:ScalarInterpolatorEx"
"http://www.cortona3d.com/source/extensions.wrl#ScalarInterpolat
orEx"
]
```

The type field defines the interpolation method. The possible values of this field are "CONSTANT", "LINEAR", "COSINE", "CUBIC" and "HERMITE". In the case of the Hermite interpolation, the params field specifies three interpolation parameters: Tension, Continuity and Bias.

Definitions of the other fields and events of the ScalarInterpolatorEx node are similar to the corresponding definitions for the standard ScalarInterpolator VRML node described at http://www.web3d.org/documents/specifications/14772/V2.0/part1/nodesRef.html#ScalarInterpolator.

## TransformSensor

```
EXTERNPROTO TransformSensor [
  exposedField     SFBool        enabled            #TRUE
  exposedField     SFBool        includeViewer      #FALSE
  eventOut   SFVec3f       translation_changed
  eventOut   SFRotation    rotation_changed
  eventOut   SFVec3f       center_changed
  eventOut   SFVec3f       scale_changed
  eventOut   SFRotation    scaleOrientation_changed
  eventOut   SFBool        transform_changed
]
[
   "urn:inet:parallelgraphics.com:cortona:TransformSensor"
"http://www.cortona3d.com/source/extensions.wrl#TransformSensor"
]
```

The TransformSensor generates events containing any transformations of the descendant geometry in the global coordinate system.
The *includeViewer* field specifies if viewer position is used in TransformSensor fields calculations or not.
The *translation_changed* event is generated if the translation is changed.
The *rotation_changed* event is generated if the rotation is changed.
The *center_changed* event is generated if the center of transform is changed.
The *scale_changed* event is generated if the scale is changed.
The *scaleOrientation_changed* event is generated if the scaleOrientation is changed.
The *transform_changed* event is generated if the transform is changed.

## ViewportSensor

ViewportSensor node returns size of 3D window in pixels.

```
EXTERNPROTO ViewportSensor [
   eventOut   SFVec2f      size_changed
]
[
   "urn:inet:parallelgraphics.com:cortona:ViewportSensor"
"http://www.cortona3d.com/source/extensions.wrl#ViewportSensor"
]
```

The *size_changed* event is generated if the size of 3D window is changed.

## X3D Nodes

### EventUtilities

These nodes allow authors to handle numerous event-types for interactive scenes without the use of the Script node.

Each node corresponds to standard VRML node. All field definitions are similar to the ISO/IEC 19775 Abstract Specification. This section provides a detailed definition of the syntax of proposed nodes.

#### *BooleanFilter*

```
EXTERNPROTO BooleanFilter [
   eventIn   SFBool set_boolean
   eventOut  SFBool inputFalse
   eventOut  SFBool inputNegate
   eventOut  SFBool inputTrue
]
[
   "urn:inet:parallelgraphics.com:cortona:BooleanFilter"
"http://www.cortona3d.com/source/extensions.wrl#BooleanFilter"
]
```

The BooleanFilter node allows routing of boolean values and negation. On receiving the *set_boolean* TRUE event, the BooleanFilter node generates the *inputTrue* event, and on receiving FALSE, it generates the *inputFalse* event. In both cases the BooleanFilter node generates the *inputNegate* event, which is the negation of the *set_boolean* value.

#### *BooleanToggle*

```
EXTERNPROTO BooleanToggle [
   eventIn       SFBool set_boolean
   exposedField  SFBool toggle     #FALSE
]
[
   "urn:inet:parallelgraphics.com:cortona:BooleanToggle"
"http://www.cortona3d.com/source/extensions.wrl#BooleanToggle"
]
```

The BooleanToggle node stores a boolean value in the *toggle* field and negates it on receiving of the *set_boolean* TRUE event.
The *set_boolean* FALSE event is ignored.

### *BooleanTrigger*

```
EXTERNPROTO BooleanTrigger [
   eventIn   SFTime set_triggerTime
   eventOut  SFBool triggerTrue
]
[
   "urn:inet:parallelgraphics.com:cortona:BooleanTrigger"
"http://www.cortona3d.com/source/extensions.wrl#BooleanTrigger"
]
```

BooleanTrigger is a node that always generates the *triggerTrue* TRUE event on receiving a *set_triggerTime* event.

### *IntegerSequencer*

```
EXTERNPROTO IntegerSequencer [
   eventIn       SFBool   next
   eventIn       SFBool   previous
   eventIn       SFBool   set_fraction
   exposedField  MFFloat  key       #[] (-inf,inf)
   exposedField  MFInt32  keyValue  #[] -1|[1,inf)
   eventOut      MFInt32  value_changed
]
[
   "urn:inet:parallelgraphics.com:cortona:IntegerSequencer"
"http://www.cortona3d.com/source/extensions.wrl#IntegerSequencer
"
]
```

The IntegerSequencer node generates the *value_changed* event on receiving a *set_fraction* event. The value of the *value_changed* event is taken from the *keyValue* array's element corresponding to the element of the *key* array the value of which equals to the value of the *set_fraction* event.

### *IntegerTrigger*

```
EXTERNPROTO IntegerTrigger [
   eventIn       SFBool   set_boolean
   exposedField  SFInt32  integerKey  #1 | (-inf,inf)
   eventOut      SFInt32  triggerValue
]
[
   "urn:inet:parallelgraphics.com:cortona:IntegerTrigger"
"http://www.cortona3d.com/source/extensions.wrl#IntegerTrigger"
]
```

On receiving a *set_boolean* event, the IntegerTrigger node generates the *triggerValue* event with the current value of *integerKey*. This is useful for connecting environmental events to the Switch node's whichChoice.

### *TimeTrigger*

```
EXTERNPROTO TimeTrigger [
   eventIn    SFBool set_boolean
   eventOut   SFTime triggerTime
]
[
   "urn:inet:parallelgraphics.com:cortona:TimeTrigger"
"http://www.cortona3d.com/source/extensions.wrl#TimeTrigger"
]
```

The *triggerTime* event is generated on receiving a *set_boolean* event. The value of *triggerTime* is the time at which *set_boolean* is received. The value of *set_boolean* is ignored.

## Inline Extension

```
EXTERNPROTO Inline [
   exposedField  SFBool      load         #TRUE
   exposedField  MFString    url          #[] [url or urn]
   exposedField  SFVec3f     bboxCenter   #0 0 0 (-inf,inf)
   exposedField  SFVec3f     bboxSize     #-1 -1 -1 [0,inf) or -1
-1 -1
]
[
   "urn:inet:parallelgraphics.com:cortona:Inline"
"http://www.cortona3d.com/source/extensions.wrl#Inline"
]
```

The *load* field defines when the Inline scene specified by the *url* field is loaded. If the *load* value is set to TRUE, the Inline scene is loaded immediately, and if its value is set to FALSE, no action is taken. The default value of the *load* field is TRUE. This means that in case of the *load* field is not specified, the Inline scene is loaded with the whole scene. The *load* field allows to load the Inline scene at any time, simply by sending TRUE event to it. Sending FALSE event to the *load* field of the already loaded Inline node unloads Inline context from the scene.