Introduction

During implementation phase of the ATG extensions, ENST has come across some limitations and would like to fix some bugs, add new features, change the way some features were added.

I. Remarks and changes on AdvancedFontStyle
I.1 Conflict with the VRML Font selection policy
I.1.1 Problem

The current spec says:
“The fontName field specifies an ordered list of desired fonts to be used, e.g. [“Times New Roman”, “SERIF”]. A second semantic permits to link a font family name to a font data stream by identifying the font name in the first string and the associated font data stream in the second string of the MFString. The syntax of the second string is:
“OD:<odid>;FSID:<fsid>”, where :
- <odid> is the numeric value of the objectDescriptorID of the associated font data stream,
- <fsid> is the numeric value of the requested font subset as conveyed by fontSubsetID within the associated font data stream.”

This is conflicting with the VRML semantics of the family field, which is to provide alternate fonts in case the previous ones are not available. Moreover, the link to the font family name is useless since the font can always be referred to by the syntax “OD:<odid>;FSID:<fsid>”.

Therefore, we suggest changing the above semantic into:
“The fontName has the same semantic as the family field of the FontStyle node. However, special fonts provided in a font data stream can be accessed using the following font name syntax:
“OD:<odid>;FSID:<fsid>”, where :
- <odid> is the numeric value of the objectDescriptorID of the associated font data stream,
- <fsid> is the numeric value of the requested font subset as conveyed by fontSubsetID within the associated font data stream.”

Consequently, the syntaxes of the FontDataDecoderConfiguration and the EnhancedFontAccessUnit need to be changed. The fontFamily and fontFamilyLength fields needs to be removed.
if (fontFormat != 0x00) {
    bit(1) storeFont;
    bit(8) fontNameLength;
    bit(8)fontName[fontNameLength];
    bit(7) fontSubsetID;
    bit(1) reserved = 1;
    bit(8) fontSpecInfo[sizeOfInstance - fontNameLength - 4];
}
}

class EnhancedFontAccessUnit() {
    bit(7) fontFormat;
    bit(1) storeFont;
    bit(8) fontNameLength;
    bit(8)fontName[fontNameLength];
    bit(7) fontSubsetID;
    bit(1) fontSubsetExtensionFlag;
    bit(8) fontSpecInfoLength;
    bit(8) fontSpecInfo[fontSpecInfoLength];
    bit(8) fontData[sizeOfInstance - fontNameLength - fontSpecInfoLength - 5]
}

I.2 Semantic of the size field

I.2.1 Problem
The FontStyle node specifies that its size field corresponds to the sum of ascent, descent and leading. But all the existing advanced typographic tools associate the size of a text with the height of the EM box.

I.2.2 Solution
We suggest to define the size field as being the height of the EM box and therefore remove the size from the sentence that says that it has the same semantics as the size field in the FontStyle node. The same applies for spacing. Its value is a multiplicative coefficient to apply to the spacing value given by the font.

I.3 Semantics of stretch, weight, fontKerning, letterSpacing, wordSpacing, baselineShift, fontVariant, featureName, featureStartOffset, featureLength and featureValue

I.3.1 Problems
The semantics of those fields in the PDAM are very unclear. For instance, what are the meaning of Condensed, Expansed?
Moreover, some fields semantics say ‘implemented according to the requirements’.

I.3.2 Solutions
We suggest either removing the fields whose semantics cannot be clarified or adding links to other specifications (CSS …), or adding text to clarify the meaning of the values.

I.4 Values of the style field

I.4.1 Problem
The allowed values in the style field are “PLAIN”, “ITALIC”, “BOLD”, “BOLDITALIC”, “UNDERLINE”, “BLINKING”, “OUTLINE”, “EMBOSS”, “ENGRAVE”, “LEFTDROPSHADOW”, “RIGHTDROPSHADOW”. But the meaning of the new values are unclear. In particular, what should be the blinking period, what ‘emboss’ and ‘engrave’ mean.

I.4.2 Solution
We suggest removing the styles “BLINKING”, “OUTLINE”, “EMBOSS”, “ENGRAVE”, “LEFTDROPSHADOW”, “RIGHTDROPSHADOW”. The blinking can be applied using color interpolation or switching, the outline using filled = FALSE. The other values should be considered to enter the ATG spec as new allowed effect
for the MatteTexture node so that it would be possible to apply these effects to any vector graphics object.

II. Changes to the Viewport node
II.1 The fit and alignment fields
II.1.1 Problem
The fit and alignment fields should be exposedField. They were made fields by mistake.
II.1.2 Solution
We suggest that the fit and alignment fields be changed from field to exposedField.

II.2 Bindability
II.2.1 Problem
The Viewport node is a bindable node but its semantics prevents it to appear anywhere in the scene.
II.2.2 Solution
We suggest to add the Viewport node to the list of bindable nodes in section 9.2.2.14. And its semantics should be change to reflect that. The spec currently says:

“A Viewport node is placed in the viewport field of a Layer2D or CompositeTexture2D node. It defines a new viewport and implicitly establishes a new local coordinate system. The bounds of the new viewport are defined by the size and position field. The new local coordinate system’s origin is at the center of the parent node in the parent’s local coordinate system.

The orientation field specifies the rotation which is applied to the viewport in the parent node’s local coordinate system with respect to the X-axis.

Viewport nodes are bindable nodes (see 9.2.2.14) and thus there exists a Viewport node stack which follows the same rules than other bindable nodes (e.g. Background2D).”

We propose to change it to:

“A Viewport node can be placed in the viewport field of a Layer2D or CompositeTexture2D node or in the scene tree as a 2D node. It defines a new viewport and implicitly establishes a new local coordinate system. The bounds of the new viewport are defined by the size and position field. The new local coordinate system’s origin is at the center of the parent node in the parent’s local coordinate system.

Viewport nodes are bindable nodes (see 9.2.2.14) and thus there exists a Viewport node stack which follows the same rules than other bindable nodes (e.g. Background2D).

The orientation field specifies the rotation which is applied to the viewport in the parent node’s local coordinate system with respect to the X-axis.”

III. Changes to the TransformMatrix2D node
III.1 Problems
The names matrix1..6 are not explicit enough for users.
III.2 Solutions
We suggest to rename matrix1..6 to $m_{xx}, m_{xy}, m_{yx}, m_{yy}, t_x, t_y$. 
IV. Modification of the XLineProperties node

IV.1 Problem

The XLineProperties node currently reuses the predefined dash pattern introduced with LineProperties node in BIFS version 1. The main issue with these predefined patterns is that dash properties are not defined in terms of size but rather in terms of high-level aspect (dash and dot combinations). While this may be enough for a few cases where dashing is not primordial, it is not enough for an author wishing to specify a given aspect of the dash pattern since the result may depend on implementations. We therefore suggest adding dash patterns to the XLineProperties node in a way that also allows for efficient line drawing, such as dash offsetting and dynamic tracing of lines. This feature is already available in the competitive SVG standard.

Another limitation of 2D BIFS is the lack of texturing for lines, which are extremely useful for simple things such as hatch filling of a line through a pixel texture node, or gradient filling through a gradient node. We suggest adding texturing info to the line properties.

Finally, we propose to change the isCenterAligned field from field to exposedField.

IV.2 Solution

We propose to modify the node as follows. Modifications are highlighted.

Replace the XLineProperties node interface by

```plaintext
XLineProperties {
  exposedField SFCOLOR lineColor 0, 0, 0
  exposedField SFINT32 lineStyle 0
  exposedField SFFloat width 1.0
  exposedField SFBOOL isScalable TRUE
  exposedField SFINT32 lineCap 0
  exposedField SFINT32 lineJoin 0
  exposedField SFFloat miterlimit 4
  exposedField SFFloat transparency 0.0
  exposedField SFBOOL isCenterAligned TRUE
  exposedField SFFloat dash_offset 0.0
  exposedField MFFloat dashes []
  exposedField SFNODE texture NULL
  exposedField SFNODE textureTransform NULL
}
```

Replace the XLineProperties semantics section by:

The XLineProperties node specifies line parameters used in 2D rendering.

The semantics of lineColor and lineStyle are the same as for the LineProperties node. But, value 6 for the lineStyle field means that dashing uses the dash information of the XLineProperties node. For the other values of lineStyle, the values of the dash_offset and dashes fields are ignored.

The dash_offset field determines the position from the start of the outline, along the outline, in the local coordinate system, where dashing begins. In case of high level primitives such as Rectangle, Circle and Ellipse, the start of the outline is not specified and therefore the use of dash_offset is undetermined. For a deterministic behavior, authors should use primitives that explicitly define the start of the outline.
The **dashes** array specifies the dash pattern. Each element is a strictly positive number expressed relatively to the width of the pen. Values at even positions in the array specify the length of drawn parts of the line. Values at odd positions in the array specify the length of non-drawn parts of the line. Dashes shall be drawn using the **lineCap** information.

The **width** field determines the width, in the local coordinate system, of rendered lines. The width is subject to scaling only when the **isScalable** field is set.

The **lineCap** field specifies the line cap style type to apply to lines. The allowed values are:

<table>
<thead>
<tr>
<th>lineCap</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>flat</td>
</tr>
<tr>
<td>1</td>
<td>round</td>
</tr>
<tr>
<td>2</td>
<td>square</td>
</tr>
<tr>
<td>3</td>
<td>triangle</td>
</tr>
</tbody>
</table>

The **lineJoin** field specifies the line join style type to apply to lines. The allowed values are:

<table>
<thead>
<tr>
<th>lineJoin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>miter</td>
</tr>
<tr>
<td>1</td>
<td>round</td>
</tr>
<tr>
<td>2</td>
<td>bevel</td>
</tr>
</tbody>
</table>

The **miterlimit** field specifies the limit on the ratio of the miter length to the line width. The value of **miterlimit** must be a number greater than or equal to 1.

The **transparency** field specifies the transparency of the outline of a shape when drawn. It supersedes the value of the transparency of a **material** node.

The **isCenterAligned** field specifies the positioning of the outline a shape. If TRUE, the line that represents the outline of the shape is drawn centered on the outline of the shape. If FALSE, the outside edge of the line that represents the outline of the shape is aligned on the outside edge of the shape.

The **texture** field, if specified, shall contain one of the various types of texture nodes. If NULL or unspecified, the line is not textured. Texture mapping coordinates are defined by the
four corners of the bounding rectangle of the outer polygon defining the shape (that is, taking into account the width of the line).

The textureTransform field, if specified, shall contain a texture transformation node (TextureTransform or TransformMatrix2D without children). If the textureTransform is NULL or unspecified, the texture is not transformed.

V. Text input for the Clipper2D node
V.1 Problem
The ATG PDAM asked for text to reflect the accepted modification of the node into a grouping node.

V.2 Input text

Node Interface

Clipper2D {
    eventIn MFNode addChildren
    eventIn MFNode removeChildren
    exposedField MFNode children []
    exposedField SFNode geometry NULL
    exposedField SFBool inside TRUE
    exposedField SFNode transform NULL
    exposedField SFBool XOR FALSE
}

Node Semantics

The Clipper2D node is a 2D grouping node that defines a free-form 2D rendering area for its children nodes. If another Clipper2D node is found in its children, children of that second clipper shall be clipped/cut using the combination of both clipping geometries, as indicated by the inside and XOR fields of both clippers.

The geometry field specifies a 2D graphical primitive to be used as the clipper shape. All 2D graphical primitives are allowed except Bitmap, PointSet2D and IndexedLineSet2D. If the geometry defines an open shape (for instance, Curve2D), the shape shall be considered as closed to perform clipping. If the geometry is NULL, children nodes are completely drawn if the inside field is FALSE, otherwise children are not drawn.

The inside field specifies whether the node shall perform a clipping operation or a cut operation. If its value is TRUE, the inside of the clipping geometry is drawn. If it is FALSE, the outside of the clipping geometry is drawn.

The transform field specifies a 2D transformation node (Transform2D or TransformMatrix2D). This node shall have no child, and is used to assign a 2D transformation to the geometry of the Clipper2D node.

The XOR field specifies whether union or intersection of this clipper with its parent clipper is made using a XOR operation or not. The XOR field is used only if this clipper and its parent clipper have the same value for the inside field, otherwise it is ignored.

Example of clipper cascade:
Let’s draw the following scene (pixel metrics, scene size 100x100):
OrderedGroup {
    children [
        Background2D {backColor 1 1 1}
        DEF Clip1 Clipper2D {


VI. Small modification of the ColorTransform node

VI.1 Problem
The node has 20 fields whose names are not explicit enough.

VI.2 Solution
We suggest to rename them as follows:

<table>
<thead>
<tr>
<th>Old name</th>
<th>New name</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix1</td>
<td>m_r</td>
</tr>
<tr>
<td>matrix2</td>
<td>m_g</td>
</tr>
<tr>
<td>matrix3</td>
<td>m_rb</td>
</tr>
<tr>
<td>matrix4</td>
<td>m_ra</td>
</tr>
<tr>
<td>matrix5</td>
<td>l_r</td>
</tr>
<tr>
<td>matrix6</td>
<td>m_gr</td>
</tr>
</tbody>
</table>
VII. Proposal of a new node: PathLayout

VII.1 Problem

While completing the Osmo4 renderer, we have (finally) integrated the layout node and started to look at the proposed modifications in the current ATG documentation. These modifications aim at providing layout of objects on an arbitrary path. However, trying to fit this functionality in the layout node is not very efficient, since layout provides only alignment notions (“BEGIN”, “FIRST”, “MIDDLE” and “END”) and is designed to place objects on several rows or columns with respect to alignment to the layout frame, when placing objects along a path doesn’t have this notion of frame nor rows/columns. Moreover, animating objects along a path needs to be more efficient than just the basic scrolling support of layout. We therefore suggest introducing a new node designed for objects-on-path positioning. The Layout node needs to be updated to remove the ability to have a path.

VII.2 Solution

We propose to define the following node:

Node Interface

PathLayout {
  eventIn MFFloat addChildren
  eventIn MFFloat removeChildren
  exposedField MFFloat children []
  exposedField SFFloat geometry NULL
  exposedField MFInt32 alignment [0, 0]
  exposedField SFFloat pathOffset 0
  exposedField SFFloat spacing 1.0
  exposedField SFFloat reverseLayout FALSE
  exposedField SFFloat wrapMode 0
  exposedField SFFloat splitText TRUE
}

Node Semantics

The PathLayout node is a grouping node used to place its 2D children along a given 2D path and possibly move them along that path. See ISO/IEC 14772-1:1998 for a description of the children, addChildren, and removeChildren fields and eventIns.

The geometry field contains a 2D geometry node defining the path. The following nodes are forbidden in that field: Rectangle, Circle and Ellipse. The path is oriented from the first point
to the last point. The length of the path is the sum of all the length of its sub-paths (set of connected curves or lines).

**pathOffset** describes the offset along the path to place the first object. Value 0 corresponds to the beginning of the path and value 1 to the end of the path. Negative values or values greater than 1 are handled according to the **wrapMode** field.

The **reverseLayout** field specifies whether the children are placed following the orientation of the path (FALSE) or the opposite orientation (TRUE).

The **alignment** field describes horizontal and vertical alignment in that order.

An object is placed as follows:
- The tangent to the path at the current position is computed, chosen with the same orientation as the path;
- The object is rotated so that the X-axis of its local coordinate system is parallel to the tangent and oriented in the same direction as the tangent;
- Alignment is applied (see below);
- The current position along the path is incremented by the current increment.

The initial position on the path is the value of the **pathOffset** field multiplied by the length of the path. The current increment depends on the current object, the next object to be placed and the alignment constraints.

**Text** nodes are considered as graphical objects if they are not direct children of the **PathLayout** node and therefore obey to the alignment constraints as specified by the **alignment** field. Otherwise, the **fontStyle** field of the **Text** node is used.

For graphical objects, alignment is applied as follows:

<table>
<thead>
<tr>
<th>alignment[0]</th>
<th>VII.3 Meaning</th>
<th>VII.4 Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ReverseLayout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>-1</td>
<td>left edge of the object is aligned with the current position</td>
<td>spacing × w&lt;sub&gt;i&lt;/sub&gt;</td>
</tr>
<tr>
<td>0</td>
<td>middle of the object is aligned with the current position</td>
<td>spacing × (w&lt;sub&gt;i&lt;/sub&gt;+w&lt;sub&gt;i+1&lt;/sub&gt;) / 2</td>
</tr>
<tr>
<td>1</td>
<td>right edge of the object is aligned with the current position</td>
<td>spacing × w&lt;sub&gt;i+1&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

where w<sub>i</sub> is the width of the current object to place and w<sub>i+1</sub> is the width of the next object to place. If there is no further object, the increment is meaningless.

<table>
<thead>
<tr>
<th>alignment[1]</th>
<th>VII.5 Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>top edge is aligned with the tangent</td>
</tr>
<tr>
<td>0</td>
<td>center is placed on the tangent</td>
</tr>
<tr>
<td>1</td>
<td>bottom edge is aligned with the tangent</td>
</tr>
</tbody>
</table>

For **Text** nodes that are direct children of the **PathLayout** node, their placement depends on the value of the **splitText** field.

If **splitText** is FALSE, the text is placed according to the **fontStyle** field of the **Text** node, with the origin of the local coordinate system being the current position on the path, and
the X-axis of that system rotated so that it is parallel to the tangent and oriented in the same
direction.

If `splitText` is TRUE, each character of the text is placed separately as if it was a
single `Text` node with the same `fontStyle` field.

The `wrapMode` field indicates action to take when the current position is less than 0 or
greater that the length of the path. The following values are defined:

<table>
<thead>
<tr>
<th>wrapMode</th>
<th>VII.6 Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The current object is not rendered, but the current position is updated as specified above.</td>
</tr>
<tr>
<td>1</td>
<td>The current position is increased or decreased by a integer number of times the path length so that it is positive and less than the length of the path.</td>
</tr>
<tr>
<td>2</td>
<td>The path is virtually extended by a tangent line at the first and last point and the current position refers to that virtual path.</td>
</tr>
</tbody>
</table>

VII.3 Modification to the Layout node

VII.3.1 scrollRate

VII.3.1.1 Problem

The `scrollRate` field of the `Layout` node currently specifies the scrolling rate in meter per seconds. This is not very convenient for authors nor well chosen in case the scene is authored in pixel metrics.

VII.3.1.2 Solution

We suggest changing the semantics so that `scrollRate` specifying the time needed in seconds to scroll the layout in the given direction. For example, a layout of 200x100 pixels scrolling vertically with a `scrollDuration` of 2 seconds will translate its objects vertically of 100/2 times the simulation frame duration in seconds (eg, 1.65 pixels at 30 fps).

VII.3.2 Scroll-Mode

VII.3.2.1 Problem

A `scrollMode` field was added to the layout node but no clear semantic was provided.

VII.3.2.2 Solution

We propose to add the following to the Layout node semantic:

Given that all the objects to be laid out are positioned, the bounding box (BB) of these objects is computed.

<table>
<thead>
<tr>
<th>scrollMode</th>
<th>VII.3.3 scrollVertical = FALSE</th>
<th>scrollVertical = TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>scrollRate &lt; 0</code></td>
<td><code>scrollRate &gt; 0</code></td>
</tr>
<tr>
<td></td>
<td><code>scrollRate &gt; 0</code></td>
<td><code>scrollRate &lt; 0</code></td>
</tr>
<tr>
<td></td>
<td><code>scrollRate &gt; 0</code></td>
<td><code>scrollRate &gt; 0</code></td>
</tr>
<tr>
<td>-1 (scroll-in)</td>
<td>Objects are initially translated so that:</td>
<td>left edge of BB is aligned with right edge of the layout frame (LF)</td>
</tr>
<tr>
<td>1 (scroll-out):</td>
<td>Objects are</td>
<td>right edge of BB is aligned</td>
</tr>
</tbody>
</table>
Value 0 of the scrollMode field corresponds to the combination of both scroll-in and scroll-out modes.

VII.3.4 the path field
VII.3.4.1 Problem
We are proposing the PathLayout node to better fulfill the functionality of this field.
VII.3.4.2 Solution
We suggest to remove this field from the Layout node.

VIII. Clarification of the Form node
VIII.1 Problem
The semantic of the Form is still unclear regarding the allowed index values and the meaning of the value 0.
VIII.2 Solution
We propose to further improve the clarification of the Form node by replacing the following paragraph:
“The constraints and the groupsIndex fields specify the list of constraints. One constraint is constituted by a constraint type from the constraints field, coupled with a set of group indices terminated by a –1 contained in the groupsIndex field. There shall be as many strings in constraints as there are –1-terminated sets in groupsIndex. The n-th constraint string shall be applied to the n-th set in the groupsIndex field.”
by
“The constraints and the groupsIndex fields specify the list of constraints. One constraint is constituted by a constraint type from the constraints field, coupled with a set of group indices terminated by a –1 contained in the groupsIndex field. There shall be as many strings in constraints as there are –1-terminated sets in groupsIndex. The n-th constraint string shall be applied to the n-th set in the groupsIndex field. A value of 0 in the groupsIndex field references the form node itself, otherwise a groupsIndex field value is a 1-based index in the group field.”

IX. Improvement of Proto and Script usage
IX.1 Problem
Accessing node fields through ECMAScript is vital to complex scene authoring. However, script access fields through their names, not through their ALL/IN/DEF indexes as the rest of the BIFS system. While this is not an issue for nodes, it is quite problematic for proto because it forces the scene to be encoded with useName in order to keep the proto field name, while the rest of the coded names (DEF nodes) is not used by the script nor the rest of the BIFS system (except when MPEG-J is present).

In a similar topic, script allows for simple creation of nodes through the node name, for example new_node = new SFNode(‘Material’). A proto may also be instantiated in this way, but this means the username encoding has to be used.

IX.2 Solution
We suggest defining a base field addressing for proto so that a script can access a proto field without having the scene configured with useName in the following way:
“A Script may access the field of a node by its name or by its field id, in other word the field index coded in ALL mode. When accessing the field by ID, the syntax used is
“_fieldN”, where N is the ALL ID. This allows accessing proto fields in a script without having to encode the scene with useName.”

We suggest allowing for proto instantiation in script through the same mechanism as field accessing:

“A Script may create a new instance of a prototyped node (eg instantiate a proto) by using the SFNode constructor with the proto name as an argument, if known, or with the syntax new_node=new SFNode(’_protoZ’) where Z is the protoID as coded in the BIFS stream”

X. Corrigendum items
X.1 FontStyle
An error was by mistake made in the corrigendum on FontStyle. The word ‘sum’ was used instead of ‘product’. Furthermore, that sentence is redundant with the VRML spec. Only the ‘Note, …’ should be kept.

The spec currently says:
“The distance between adjacent text baselines is the sum of the size and the spacing. (Note, that this makes that the text size is the sum of the ascent + descent + leading, the latter which is the interline spacing, the logical amount of space to be reserved between the descent of one line of text and the ascent of the next line).”

It should be changed into:
“The text size is the sum of the ascent + descent + leading, the latter which is the interline spacing, the logical amount of space to be reserved between the descent of one line of text and the ascent of the next line.”

X.2 InputSensor
There is a discrepancy between the spec and the reference software regarding the InputSensor node. The url field is an SFString in the spec but an MFString in the reference software. As all urls, it should be an MFString.