Abstract:

We have proposed and implemented function-based extensions of Extensible 3D (X3D) and its predecessor Virtual Reality Modeling Language (VRML) which allow for defining time-dependent geometric shapes, their appearances and transformations with analytically defined parametric, implicit and explicit functions. The function-defined shapes can be used together with the standard X3D and VRML shapes. In this paper we present the most recent significantly revised and amended definition of the FVRML/FX3D nodes. Besides defining shapes by analytical functions, we have developed interactive function-based shape modeling tools. We have also extended these interactive shape modeling tools to work on the Grid.

1. Introduction

1.1 Web-based visualization

3D web visualization methods, which can be used when creating cyberworlds, can be categorized into three classes: image transmission, raw data transmission, and model transmission.

Image transmission assumes that 3D web visualization can be implemented by transmitting to the clients 2D images rendered at the servers. Raw data transmission assumes that geometric models reside and are rendered at a client computer, while only the necessary data (e.g. object coordinates, object status, events, etc.) is transmitted over the Internet to control the rendering and interaction. When model transmission is used, 3D objects are rendered at the client computers, but the models are transmitted over the Internet together with other communication data. Whenever updating or creating of new models is required, the respective data can be seamlessly delivered to the client computers and cached there. With the modern computers and web visualization software, quite high visual realism can be achieved with this approach. Many web based and client-server based virtual environments use this scheme (e.g., Project Entropia http://www.project-entropia.com and Second Life http://secondlife.com), Extensible 3D (X3D) and its predecessor Virtual Reality Modeling Language (VRML) also use this scheme.

1.2 VRML and X3D

Virtual Reality Modeling Language was proposed and adopted as International Standard in 1997. The Extensible 3D (X3D) specification is the successor to VRML97, which provides a backward compatibility with VRML97.

VRML is capable of representing static, animated and interactive 3D virtual environments by scripts and events. However, if complex shapes are used, the VRML file may become large, and it will take a long time to download it before the scene can be viewed. Even with the gzip compression, the size of a VRML scene can be of several megabytes including image textures. Second, the standard VRML does not support such advanced visualization techniques as Bump Mapping, Environmental Mapping, etc. Third, there is no native constructive solid geometry support in VRML, and it only can be done using commercial software through import and export functions at the level of polygon meshes. Finally, the animation support existing in VRML is quite limited. Many solutions were proposed to overcome these deficiencies. In [1], morph node as well as some supporting nodes were presented to interpolate smooth animated 3D shapes. An extension for visualizing volumetric data is proposed in [2]. GeoVRML (http://www.geovrml.org) is an extension for defining and visualizing large scale geographic data.

There are several VRML browsers, which provide enhanced features for VRML. Cortona developed by ParallelGraphics, and Blaxxun’s blaxxun Contact provide many extensions including advanced rendering techniques, such as Bump Mapping and Environmental Mapping. These extensions partially remove the limitation of VRML. Bitmanagement’s BS Contact VRML/X3D viewer provides many advanced visualization features. Besides NURBS, Bump Mapping, and Environmental Mapping, other new techniques including particle systems, multitexturing, movie textures, and texture composition are implemented there as extensions. The most important improvement in the BS Contact VRML/X3D are the programmable shaders which can make procedurally generated textures, volume textures, real-time shadows, displacement mappings, and other special visual effects.

The Extensible 3D (X3D) is a new open standard for real-time communication across various applications and networks. It includes features used in engineering and scientific visualization, CAD, multimedia and entertainment, etc. Though it is still an international standard draft, X3D has been already widely accepted as the successor of VRML. It introduces many new features to enhance the VRML meanwhile still provides backward
In terms of shape modeling features, X3D has many benefits, and has already been widely used in many industrial areas. However, it still has several deficiencies. First, though it provides advanced polygon-based representation methods such as triangle fans and strips, as well as compressed binary encoding, the file sizes would still be large for complex shapes. Second, without commercial modeling software, constructive solid geometry (CSG) is not supported by X3D. Even with commercial software, CSG can only be done at the level of polygon meshes. Third, it is not easy for authors to apply 3D geometric textures to a 3D geometry. Finally, the animation support in X3D is still limited, despite some improvements which have been done. We solve these problems by proposing the function-based extension of X3D.

In this paper we discuss how to greatly improve the shape modeling abilities of VRML and X3D by using the hybrid function-based shape modeling approach [7-10]. This approach allows for extending VRML and X3D with practically any type of object’s geometry, sophisticated graphics appearance and transformation by defining them analytically or by scripts straight in the VRML/X3D code. In contrast to the polygon-based models of VRML/X3D, their function-based extensions FVRML/FX3D allow for a great reduction of the model sizes and provide an unlimited level of detail. Also, these extensions introduce to VRML/X3D set-theoretic as well as any other analytically-defined operations.

In this paper we present the most recent significantly revised and amended definition of the FVRML/FX3D nodes.

2. Hybrid function-based shape modeling in VRML and X3D

For defining geometry, appearance and their transformations, three types of analytical functions can be used concurrently. They are implicit, explicit and parametric functions which can be used as individual analytical formulas as well as grouped into function scripts similar to VRMLScripts (JavaScripts) which are used in X3D and VRML.

Implicit functions are \( f(x, y, z, t) = 0 \), where \( x, y, z \) are Cartesian coordinates and \( t \) is the time. The function equals to zero for the points located on the surface of a shape.

Explicit functions are \( g = f(x, y, z, t) \). The function value \( g \) evaluated at any point of the 3D space can be used either as an argument of other functions (e.g. for defining colors or parameters of transformations) or as an indicator of the sampled point location. Hence, explicit functions can be used for defining bounded solid objects. In this case, the function equals to zero for the points located on the surface of a shape. Positive values indicate points inside the shape and negative values are for the points outside the shape.

Parametric functions define surfaces and solid objects as:

\[ x = f_1(u, v, w, t); \quad y = f_2(u, v, w, t); \quad z = f_3(u, v, w, t) \]

where \( x, y, z \) are Cartesian coordinates of the points, while \( u, v \) and \( w \) are parametric coordinates, and \( t \) is the time. Parametric functions can also define \( r, g, b \) values of color or parameters of transformations.

Geometry, 3D texture, and color can be defined by implicit, explicit or parametric functions in their own domains and then merged together into one shape by the predefined mapping functions. When modeling shape, first a geometric texture is mapped onto the original shape’s geometry, and then the color is mapped to the textured shape to form the final shape. This process repeats as many times as needed.

For example, we can define the original shape by the following implicit function:

\[ -((2u^2 + x^2 + y^2 - 1)^3 - (0.1z^2 + x^2y^3)) & (-(x+z)) = 0 \]

where sign & stands for intersection.

Then, we create a geometric texture on this shape by displacing each point \( p \) along the normal \( N \) to the surface as follows:

\[ p = p + g \cdot N \]

where displacement \( g \) is defined by the following explicit function:

\[ g = 0.02(\sin(2\pi x)\sin(2\pi y) + \sin(2\pi x)\sin(2\pi z) + \sin(2\pi y)\sin(2\pi z)) \]

Finally a parametrically-defined color is applied onto the textured shape by mapping the color plane onto the shape:

\[ r = 1; \quad g = \text{fabs} \sin(u); \quad b = 0; \]

The final shape created with these implicit, explicit and parametric functions and the FVRML code are shown in Figure 1.

![Texture Shape](image1)

![Color Shape](image2)

FVRML prototypes are omitted to save space.
3. FVRML and FX3D

FVRML/FX3D offer six additional nodes, which are FShape, FGeometry, FAppearance, FMaterial, FTexture3D and FTransform. These nodes can be used together as well as with the standard VRML/X3D nodes (Figure 2).

FShape node is a container similar to the VRML/X3D’s Shape node. It contains FGeometry or any standard geometry node, and FAppearance or the standard Appearance node. These nodes define the geometry and the appearance of the shape, respectively. FShape may be called from FTransform node or from the standard Transform node.

FGeometry node is designed to define a geometry using implicit, explicit or parametric functions. It contains either one string, or a script defining a function representing the geometry, or a URL to the function definition.

FAppearance node may contain FMaterial or the standard Material, the standard Texture node, and FTexture3D node. FTexture3D defines geometric textures while the standard Texture node is used for image texture mapping.

FMaterial node defines diffuse, specular, and emissive colors, as well as transparency, ambient intensity, and shininess with parametric and explicit functions. Parametric functions directly define color values \( r \), \( g \), \( b \). When the color is defined with an explicit function, its values are linearly mapped to the \( r \), \( g \), \( b \) values of the actual color with a user-defined color interpolation map.

FTexture3D node defines 3D geometric textures using explicit and parametric functions. These functions define displacements of the original shape’s geometry defined in FGeometry or in any standard geometry node. The functions are defined either in one string, or in a function script, or via a URL to the function definition.

If the standard Geometry node is used in place of FGeometry node, the standard shapes of VRML/X3D will be assigned an appearance defined in FAppearance node. This ability of using the function-based nodes with the standard geometry and appearance nodes of VRML/X3D adds many new features to VRML/X3D, some of which are illustrated in this section.

FTransform node contains function-defined operations. It may contain FShape and other FTransform nodes as its children. FTransform contains either one string, or a script defining a function representing the operation, or a URL to the function definition as well as other fields to preserve the consistency with the standard Transform node. Both, set-theoretic and affine transformations can be defined as functions of time in FTransform node. Ste-theoretic operations are defined as in [11].

In Appendix in Table 1, several examples of defining shapes in FVRML and FX3D are given while Table 2 contains the description of the node fields.

The developed FVRML/FX3D extension is being used for teaching computer graphics at several universities. This is a part of the computer graphics assignment “Implicit Fantasies and Parametric Metamorphoses in Cyberworlds” where the students have to design a shared VRML environment containing animated function-defined objects. Examples of student works are shown in Figure 3.

The FX3D/FVRML plug-in for blaxxun Contact, Bitmanagement BS Contact VRML/X3D and Xj3D can be downloaded from the project web page.

---

**Figure 1.** Modeling shape by consecutive definition of its geometry, texture and color

**Figure 2.** Using extended and standard nodes of VRML/X3D. The standard nodes are shaded

```xml
FShape
  FGeometry
    definition "(1-x^2-y^2-z^2)&(1-x-2*y-z)&(1+x-2*y-z) & (1-x-2*y+z)&(1+x-2*y+z)"
    bboxSize 3 3 3
    resolution [50 50 50] }
  FAppearance
    appearance FAppearance
      texture3D FTexture3D { 
        definition "0.02*(sin(2*pi*x)*sin(2*pi*y)+sin(2*pi*x)*sin(2*pi*z)) +2*pi*y)*sin(2*pi*z)"
      }
      material FMaterial { 
        diffuseColor "r=1;  g=fabs(sin(u));  b=0;" 
        emissiveColor 0 0 0 
        specularColor 1 1 1 
        shininess 0.7 
      }
    }
  }
```
4. Implementation issues

4.1 General considerations

There are two general ways of introducing new nodes to VRML/X3D: browser-specific customization and VRML/X3D script node. Browser-specific customization can be done by developing an additional library using the software development kit (SDK) from the browser’s developer. The additional library can be highly integrated into the browser, and sometimes can even act as a part of the browser, which results in advance performance. Moreover, since the library is entirely integrated into the browser, nodes in the extensions can be used directly as standard nodes without declaring any prototypes.

VRML/X3D script node can be implemented through the script node by using a scripting interface provided by VRML and X3D. The VRML/X3D-compliant browsers must support both ECMA-script, (standard scripting language http://www.ecma-international.org/publications/standards/Ecma-262.htm) and Java class files. Therefore, to extend VRML/X3D with new nodes the implementation can build prototypes in order to encapsulate scripts using the script nodes. Since both JavaScript and Java class files are platform-independent, the implementations can use any browser on any platform. Another advantage is that no software other than the Java runtime environment (JRE) has to be downloaded and installed and this is commonly pre-installed on personal computers. When using the script node and the prototype mechanism, the prototypes, as well as the scripts, are downloaded automatically from the web by the browser. The whole process is transparent to the viewers. Even for browsers that do not fully support Java-based scripts, there will be no error messages and the extension nodes will be simply ignored.

Both implementations are described in 4.2 and 4.3.

For the both implementation methods, to visualize the function-defined objects in VRML and X3D, we have to convert them into the standard VRML and X3D primitive objects (points, lines and polygons), which will be then rendered by the browser. In addition, this process should not block the user’s interaction with the standard objects. The visualization pipeline illustrated in Figure 4 includes eight different modules sharing two threads, namely: Parser, Compiler, TimeGenerator, ShapeGenerator, TextureGenerator, ColorGenerator, TransformOperator, and PrimitiveGenerator. The diagram shows how function definitions can cooperate with VRML and X3D browsers. Solid lines represent the execution flow and dotted lines represent the communication between the threads. One thread is responsible for parsing and compiling the source code into an intermediate code suitable for evaluation, while the other thread is responsible for conversion of the hybrid function-defined models into primitive objects. For each specific frame, one set of the standard objects is generated on-the-fly.
In the visualization pipeline, Parser, Compiler, ShapeGenerator, TextureGenerator, ColorGenerator, TransformOperator and PrimitiveGenerator (shown as grey boxes) are pluggable, since the functionality of these modules may need to be changed. The TimeGenerator module is still hard-coded in the implementation, since it has a fixed task of generating time signals synchronously with the real time. Both the function scripts and the analytical formulae are compiled using an embedded compiler when they are loaded or changed. They can be executed and return the function values whenever the function evaluations are required.

4.2 Browser specific native script implementation

Blaxxun Contact (http://www.blaxxun.com) and Bitmanagement BS Contact VRML/X3D (http://www.bitmanagement.de) do not support Java scripting interface. Instead, they support a special kind of script called native script. By writing a specially constructed string in the url field of the script node, this browser can automatically load a COM library and execute the functions in the library. The library can be written in C++, Delphi, Visual Basic and other programming languages that can generate native codes running on the machine, hence providing the best performance. Based on these requirements, FVRML extension for Blaxxun Contact and FVRML/FX3D for BS Contact VRML/X3D browsers were built using C++ programming language. The parser module is implemented using YACC for C++.

4.3 Java-based VRML/X3D script implementation

Ideally, all certified VRML/X3D browsers should be able to support the function-based extension. All certified browsers should support both ECMAScript and Java in its script node. These two programming languages are platform-independent. For Java, the binary class file is platform-independent too. Nevertheless, due to the complexity of the extension, ECMA Script is not suitable for this implementation. Other programming bindings may be supported by certain browsers, such as C++, and may have a better performance. However, since they are not fully standardized and not platform-independent in binary form (executables or libraries), they are not considered a choice. For these reasons, we chose the Java programming language for the development. As the interface of the script is fully standardized, for this implementation we selected the Xj3D browser (http://www.xj3d.org) as the development environment. The extension developed in this environment can be used by other certified browsers without any modification, even in binary form (i.e. class files) without source codes. The parser module is implemented using YACC for Java.

4.4 Optimization of the visualization

When computationally expensive and especially time-dependent (animated) function models are used, the visualization can be slowed down by interpreting the intermediate code on a virtual machine when evaluating the functions. One way to speed up the process is to optimize the intermediate code. However, this optimization would be difficult to implement, and it can only gain little in performance improvement. An alternative is to employ the just-in-time (JIT) compiling technique [12]. This dynamically converts the intermediate code into a high-performance binary code, which is then executed by the computer. The optimized binary code significantly improves the visualization performance.

The generation of numerous primitive objects constituting time-dependent objects requires very fast algorithms for making these primitive objects on-the-fly. However, when computationally expensive mathematical functions and scripts are used, caching of the meshes appears to be feasible. We implemented caching as an option when visualizing animated function-defined shapes.

5. Interactive shape modeling

The described way of modeling shapes with analytical functions and scripts is quite versatile but it may be difficult for many users to model something meaningful just by direct typing of mathematical formulas in FVRML/FX3D code. Therefore, we have also developed an interactive modeling tool which allows for creating shapes by gradual interactive modifications, such as sculpting, carving and painting. The initial shape can be either selected from one of the basic shapes, or defined by an analytical implicit function (Figure 5). The resulting shape model contains information of all the modifications applied to the geometry and appearance of the shape. The shape can be moved in three dimensions. Any level of detail can be defined, which changes the size of the triangles used for polygonization and re-renders the shape accordingly (Figure 6a). The shapes are then saved in the Base64-encoded binary form of FVRML format, which can be visualized by the FVRML plug-in implemented as described in 4.2 (Figure 6b). We have also extended these interactive shape modeling tools to work on the Grid [13].
Fig. 5. Definition of the initial shape by an implicit function

There is also a web-based version of the collaborative shape modeling tool available at [http://www.ntu.edu.sg/home/assourin/FVRML.htm#Collaborative](http://www.ntu.edu.sg/home/assourin/FVRML.htm#Collaborative). This application allows several users to participate in the design of an FVRML scene. This tool is consistent with the paradigm of the hybrid function-based modelling. It lets the users introduce any object or operation by analytical functions (geometry or appearance), interactively combine them into another function-defined object, discuss the design with other users and save the created object into VRML data file for further use (Figure 7).

Fig. 7. Collaborative shape modeling on the web

Conclusion

In this paper, we describe the most recent version of the function-based extension of VRML and X3D that allows for defining shapes by analytical implicit, explicit and parametric functions. These functions define the shape’s geometry, 3D geometric texture, color and their transformations through time. The functions can be used concurrently and grouped into scripts. Custom time-dependent operations can also be defined by analytical functions. With the proposed extension, programming of complex shapes becomes very easy in VRML and X3D. Sophisticated shapes can be defined by typing the code or interactively without increasing the file size, as normally happens when polygon-based models are used. The developed FVRML/FX3D extension has successfully been used in teaching computer graphics and shape modeling. The FVRML/FX3D plug-in is...
References


Appendix

Table 1  Examples of FVRML and FX3D shapes

<table>
<thead>
<tr>
<th>Transform</th>
<th>Function-defined appearance of the VRML shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>FShape {</td>
</tr>
<tr>
<td>loop TRUE</td>
<td>appearance FAppearance { material FMaterial {</td>
</tr>
<tr>
<td></td>
<td>timeSpan -3.1415926 3.1415926</td>
</tr>
<tr>
<td></td>
<td>diffuseColor &quot;sin(sqrt(x<em>x+y</em>y+z*z) * pi + t)&quot;</td>
</tr>
<tr>
<td></td>
<td>patternColor [0 1 0 1 0 0] patternKey [-1 1 ]</td>
</tr>
<tr>
<td></td>
<td>texture2D FTexture2D { timeSpan 0.0 0.12 }</td>
</tr>
<tr>
<td></td>
<td>definition &quot;</td>
</tr>
<tr>
<td></td>
<td>falts(t-0.03)<em>(sin(4</em>pi<em>x)<em>sin(4</em>pi</em>y))</td>
</tr>
<tr>
<td></td>
<td>+sin(4<em>pi</em>x)<em>sin(4</em>pi<em>z)+sin(4</em>pi<em>y)<em>sin(4</em>pi</em>z)&quot;</td>
</tr>
<tr>
<td></td>
<td>Geometry IndexedFaceSet { coord Coordinate [ . . . ] }</td>
</tr>
</tbody>
</table>
|          | } } } } } } } } } } } } } } } } } } } } } } )

| <ProtoInstance name="FSShape">       |
| <fieldValue name="cycleInterval" value="5"/> |
| <fieldValue name="loop" value="true"/> |
| <fieldValue name="appearance"/>       |
| <ProtoInstance name="FAppearance">   |
| <fieldValue name="material"/>         |
| <ProtoInstance name="FMaterial"/>    |
| <fieldValue name="timeSpan" value="-3.14 3.14"/> |
| <fieldValue name="diffuseColor" value="sin(sqrt(x*x+y*y+z*z) * pi + t)"/> |
| <fieldValue name="patternColor" value="0 1 0 1 0 0"/> |
| <fieldValue name="patternKey" value="[-1 1 ]"/> |
| <fieldValue name="geometry"/>         |
| <ProtoInstance name="FGeometry"/>    |
| <fieldValue name="resolution" value="800 600"/> |
| <fieldValue name="parameters" value="x:1 1 1 y:1 1 1 z:1 1 1"/> |
| <fieldValue name="timeSpan" value="0.01 1"/> |
| <fieldValue name="definition" value="" |
| function parametric_x(u,v,w,t) |
| {u=(u+1)*pi*2*t;v=(v+1)*pi; |
| return 0.1*cos(v)*cos(u)+0.5*cos(u)*(1+0.4*cos(1.5*u));} |
| function parametric_y(u,v,w,t) |
| {u=(u+1)*pi*2*t;v=(v+1)*pi; |
| return 0.1*sin(v)+0.4*sin(1.5*u)-1; |
| function parametric_z(u,v,w,t) |
| {u=(u+1)*pi*2*t;v=(v+1)*pi; |
| return 0.1*cos(v)*sin(u)+0.5*sin(u)*(1+0.4*cos(1.5*u)-1);} |
| <ProtoInstance/> </ProtoInstance> <fieldValue> |

| Inline       | <url "head.wrl" /> |
| FShape {appearance Appearance { material Material |
| diffuseColor 0.1 0.1 0.1} } |
Table 2  Meaning of the FVRML/FX3D fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(u,v,w,t); y(u,v,w,t); z(u,v,w,t);</td>
<td>In this field analytical definition of the shape's geometry or geometric texture has to be defined. It can be given either as individual implicit or parametric formulas or as a function script. The names of Cartesian coordinates is to be used. For defining surfaces and solid objects, x, y, z variables are to be used. For defining parametric surfaces, two of the parametric coordinates are to be used (e.g., u and v). For defining curves, only one of the parametric coordinates is to be used. New variable names can be introduced in function scripts as it is normally done in JavaScript. The formulas can be set-theoretic operations that are implemented with functions (as in Chapter 3). If the continuity equals to 0, the set-theoretic operations are implemented with min/max functions (as in Chapter 3). If</td>
</tr>
<tr>
<td>function x(u,v,w,t); y(u,v,w,t); z(u,v,w,t);</td>
<td></td>
</tr>
<tr>
<td>function script; return x(u,v,w,t); y(u,v,w,t); z(u,v,w,t);</td>
<td></td>
</tr>
<tr>
<td>function parametric_x(u,v,w,t); function script; return x(u,v,w,t);</td>
<td></td>
</tr>
<tr>
<td>function parametric_y(u,v,w,t); function script; return y(u,v,w,t);</td>
<td></td>
</tr>
<tr>
<td>function parametric_z(u,v,w,t); function script; return z(u,v,w,t);</td>
<td></td>
</tr>
<tr>
<td>parameters [-1 1 -1 -1 1]</td>
<td>Default: parameters [-1 1 -1 -1 1]</td>
</tr>
</tbody>
</table>
the continuity is greater than 0, they are implemented with the so-called R-functions.

```
continuity 0
```

**bboxCenter x y z**
Default: bboxCenter 0 0 0

It defines the origin of the coordinate system of the implicitly/explicitly defined surface.

```
bboxSize x y z
Default: bboxSize 10 10 10
```

It defines the size of the bounding box, and thus the range of variables x, y, z for the implicitly/explicitly defined shape.

```
resolution [x y z]
Default: resolution [50 50 50]
```

It defines the rendering resolution. There must be 3 values for solid shapes, 2 values for parametric surfaces, and 1 value for parametric curves.

```
diffuseColor "frep(x,y,z,t)" or diffuseColor "frep(x,y,z,t)"
function script; return frep(x,y,z,t);"
```

In this field the function definition of the shape's diffuse color is to be given. It can be typed either as individual explicit or parametric formulas or as a function script. Cartesian coordinates are x, y, z, while the names of parametric coordinates are u, v, w. Ranges of the variables can be defined in parameters, bboxCenter, and bboxSize fields. Parameter t is the time. The formulas can be also called by a URL. In case if the color is defined by an explicit function, color interpolation will be done using the values defined in fields patternKey and patternColor.

```
emissiveColor "frep(t)"
```

It defines a time-dependent emissive color by an explicit function or script. In case of the color definition by one explicit function, color interpolation will be done by the current pattern. Parameter t is the time. The formulas can be also called by a URL. The emissive color is assigned to the whole shape and cannot vary across it. In case if the color is defined by an explicit function, color interpolation will be done using the values defined in fields patternKey and patternColor.

```
specularColor "frep(t)" or specularColor "frep(t)"
```

It defines a time-dependent specular color by an explicit function or script. In case of the color definition by one explicit function, color interpolation will be done by the current pattern. Parameter t is the time. The formulas can be also called by a URL. The specular color is assigned to the whole shape and cannot vary across it. In case if the color is defined by an explicit function, color interpolation will be done using the values defined in fields patternKey and patternColor.

```
ambientIntensity "frep(t)"
```

It defines a time-dependent ambient intensity by an explicit function or a script. The ambient intensity is assigned to the whole shape.

```
shininess "frep(t)"
```

It defines a time-dependent shininess by an explicit function or a script. The shininess is assigned to the whole shape.

```
transparency "frep(t)"
```

It defines a time-dependent transparency by an explicit function or a script. The transparency is assigned to the whole shape.

```
patternKey [a1 a2 a3 ... an]
```

It defines the values of the explicit diffuseColor, emissiveColor or specularColor function or a script which will be used for mapping to patternColor.

```
patternColor [r1 g1 b1 r2 g2 b2 ... r_n g_n b_n]
```

It defines the colors which will be used for interpolating the color defined by an explicit diffuseColor, emissiveColor or specularColor function or a script based on the current patternKey. Linear interpolation of colors will be done.

```
g=...; patternValue(g);
```

The function is used in scripts to generate colors defined by patternKey and patternColor.

```
translation "x=x(t); y=y(t); z=z(t)"
```

It defines translation parameters by explicit functions or scripts.

```
scale "x=x(t); y=y(t); z=z(t)"
```

It defines scaling parameters by explicit functions or scripts.

```
rotation "x=x(t); y=y(t); z=z(t); a=a(t)"
```

It defines rotation parameters by explicit functions or scripts.

```
center "x=x(t); y=y(t); z=z(t)"
```

It specifies a function-defined translation offset from the origin of the local coordinate system (0, 0, 0).

```
scaleOrientation "x=x(t); y=y(t); z=z(t); a=a(t)"
```

It specifies a function-defined rotation of the coordinate system before the shape is defined. In this field either a name of the predefined set-theoretic operation or an individual formula or a script defining a set-theoretic operation on shapes defined by functions f(x,y,z,t) and g(x,y,z,t) have to be typed. The respective FSshapes are to be defined as children of FTransform node. Set-theoretic operations do not apply to parametrically defined shapes.

```
timeSpan t1 t2
Default: timeSpan 0 1
```

It defines a time interval of variable t when defining a time-dependent process (shape's geometry, color, texture or transformation). Variables t1 and t2 can take any real values including negative.

```
startT ime t
Default: startT ime 0
```

It defines the values of time to start a time-dependent process.

```
stopTime t
Default: stopTime 0
```

It defines the values of time to stop a time-dependent process.

```
cycleInterval t
Default: cycleInterval 1
```

It defines the duration of the time-dependent process cycle in sec.

```
frames n
Default: frames 0
```

It defines the number of frames to be cached. If 0 is defined, rendering time-dependent processes will be performed on the fly.

```
loop TRUE/FALSE
Default: loop FALSE
```

It enables or disables looping of time-dependent processes.

```
enabled TRUE/FALSE
```

It enables or disables time-dependent processes.