Function-defined shape node for VRML

F.M. Lai[1] and A. Sourin[2]

[1]Creative Technology Ltd., Singapore [2]Nanyang Technological University, Singapore

Abstract

This paper describes how the web-based visualisation can be greatly improved using the function-based shape modelling technique. We propose the function-defined VRML shape node, which allows the content creators to describe any complex models with relatively small functions compared to the large-size polygonal mesh based VRML nodes. The design, the implementation details, and the application examples of the proposed node are discussed. The software is available for downloading from the project website.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling; I.6.5 [Simulation and Modeling]: Model Development

1. Web-based visualisation

Web-based visualisation services first appeared when some platform-independent programming languages and data exchange formats like HTML, VRML and Java were freely distributed.

A variety of web-based visualisation systems have been developed. Progressive reconstruction and isosurface transmission suggested by Engel et al[1] aimed to reduce the amount of data to be reconstructed and transmitted during visualisation. In order to improve the progressive reconstruction approach, Engel[2] has demonstrated the construction of stripped surface representations and adaptive hierarchical concepts used to minimize the number of vertices that have to be reconstructed, transmitted, and rendered. Seidel et al[3] have presented a framework to acquire high quality 3D models of real world objects including both geometry and appearance information presented by textures, bump maps or bi-directional reflectance distribution function used to describe the way a surface reflects light. Fogel et al[4] have proposed a web architecture for progressive delivery of a 3D content, which is based on a progressive compression representation integrated into X3D framework. Taubin et al[5] have proposed another popular approach to delivering a 3D content over the Internet within a reasonable time, besides introducing a new adaptive refinement scheme[6] for storing and transmitting the manifold triangular meshes in progressive and highly compressed form. Recently, Pajarola and Rossignac[7] have proposed a compressed progressive meshes approach, which uses a new technique to refine the topology of the mesh in batches. Among these services, the polygon mesh representation is often used to represent geometric models. Thus the most popular 3D web content format Virtual Reality Modelling Language (VRML)[8] uses this representation for complex objects.

Although there are many workarounds to improve the overall experience in 3D web visualisation, the mainstream of the current web visualisation is based on the scenario where the publisher creates a 3D model rather than an image and sends it to the viewer, which then renders and manipulates the model. As the rendering is done at the client's side, the smoothness of presenting a complicated scene, that is formed by millions of polygons, is much dependent on the Internet bandwidth. In many cases, the quality of the scene has to be compromised so that the 3D modelling file size is kept within an optimised range. From the different web visualisation models or techniques available today, some techniques are adopted to improve the performance of visualisation; while some are used to transmit the visualisation data interactively based on the level of details and the network load. However, the root of the problem is the actual representation of the visualisation data to be transmitted over the network. The compactness of the visual representation directly affects the overall user experience in interactive web-based visualisation.

In this project, we design and implement a generic function-defined shape node for VRML. In Section 2, the function-based approach to shape modelling in web visualisation is addressed. In Section 3, the proposed VRML node for a generic function model is introduced. In Section 4, we describe an implementation and test results for the function-defined shape node developed for the particular function model F-Rep with its dedicated description language HyperFun. In the last section, further issues are discussed and conclusions are made.

2. Function-based shape modelling over the Internet

2.1. Function-based shape modelling

Function-based shape modelling is becoming increasingly popular in computer graphics. The idea of function-based approach to shape modelling is that complex geometric shapes can be produced from a "small formula" rather than thousands of polygons. Usually, parametric or implicit functions and their modifications are used to define the shapes. For rendering such function-defined shapes, either ray tracing or polygonisation followed by fast polygon rendering is used. Alternatively, the function-defined shapes can be voxelised and rendered as a set of points. It must be admitted, though, that function defined models may sometimes suffer a serious problem which is large time needed to evaluate the defining functions. Nevertheless, many works have been done in the direction of accelerating the function evaluation when performing their rendering, and now many function-based models, which existed rather theoretically just a few years ago, revolutionise the ways of shape representation. In the next subsections, we select the direction of expanding function-based modelling to web visualisation, overview the existing projects attempting such a visualisation, and propose our own solution to this problem.

2.2. VRML and its existing extensions

As it was mentioned in the previous section, the 3D web content can be published using various visualisation techniques. Among them, VRML is the most common format adopted to define 3D objects in the virtual web spaces.

In this project, we have chosen VRML to integrate with function-based modelling technique for web visualisation. Besides its popularity in defining 3D web content, VRML suffers an obvious performance drop whenever it is used to describe complex scenes. This drawback directly encourages designing a new VRML shape node based on the function-based approach.

A generic function-based geometric node can be defined in VRML since this language is highly extensible. There are various VRML extensions that have been implemented successfully. For example, Alexa et al[9] have suggested the Morph node that helps to interpolate among several geometries. Another example is customized GeoVRML[10] nodes that are defined to provide a suite of solutions for representing and visualizing geographic data using a standard VRML97 browser. Wyvil and Guy[11] have introduced a new VRML extension for skeletal implicit surfaces with a limited set of operations like blending, warping and Boolean operations. Besides that, Pittet et al[12] have proposed a new real-time Isosurfacing node. The proposed node, which is based on the so-called Marching Cubes algorithm[13], allows real time rendering of an isosurface from 3D source data. Grahn et al[14] have introduced trimmed NURBS in VRML. This adopted approach allows visualisation of complex CAD models in VRML. Meanwhile, Ginis and Nadeau[15] have presented new VRML extensions for scientific visualisations. The suggested nodes are not mere collections of visually complicated parts, but can offer significant help in simplifying a user's job when visualizing vector fields.

2.3. Function-defined geometric node

In this project, a generic function-based node for VRML is proposed as a plug-in to a VRML browser. The idea of introducing such a node is to replace large polygonal representations of complex shapes with small formulae, which will be transmitted through the network much faster and therefore will improve the efficiency of web visualisation.

The proposed architecture has obvious improvements as compared to the conventional VRML architecture. When a client tries to view a complex function-defined geometric model via the Internet, the VRML browser plug-in sends a view request to the remote content server. A small VRML file with a reference to its respective function-defined model description is sent back to the client. The size of the VRML file is small since the complex geometric model is described using little functions. When the VRML data reaches the client machine, the VRML plug-in starts to parse the VRML immediately. The scene graph traversal module proceeds and renders the scenes during the traversal. Once the scene graph traversal module finds the customized function-defined node, the browser plug-in will download the model description according to the specified URL. Alternatively, the description can be embedded in the VRML code. The polygonisation of the function model is completed at the client machine, so that no large amount of rendering data is transferred across the network. Immediately after the polygonisation has completed, the complex geometric model will be presented to the client.

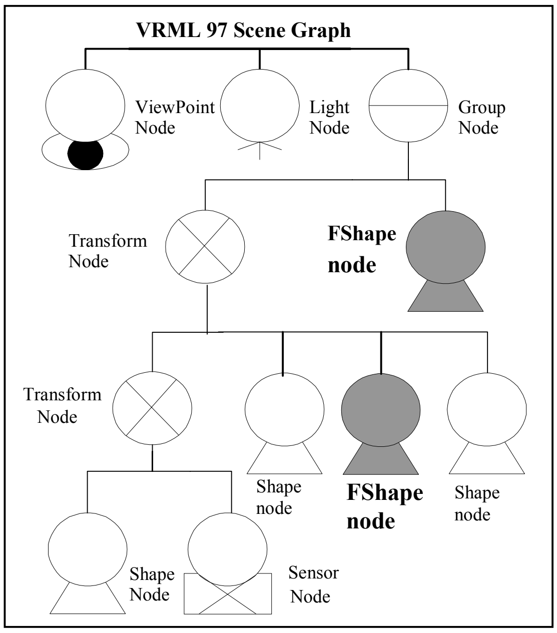

Figure 1: *Scene graph with FShape node.*

With the proposed architecture design, a new function-based geometric node, called *FShape*, has been proposed for VRML, as shown in Figure 1. The *FShape* node is designed in such a way that it is able to take in different function-defined models without much integration effort once the parsing and polygonisation modules are available. Alternatively, if the model requires special parser and polygoniser, they can be developed by the user and integrated with the node.

3. FShape node implementation

A VRML node extension can be achieved either via plug-in customisation or through VRML EAI interface[16]. For plug-in customisation, the overall performance of the extended node will be much better than the other one, since the module is implemented with a low level programming language. The performance improvement becomes obvious when involving complex geometric models, which may require complex computations and algorithm evaluations. Although the plug-in customisation may closely tie the node extension with the supported VRML browser plug-ins, the overall performance may need to be considered first if the node extension involves heavy computation processes.

Another possible approach to implement the extension is via a VRML Script node and its EAI interface. The implementation using JavaScript or Java may allow extension across different browsers, however this approach may have serious performance setback. The polygonisation implemented using JavaScript language will only be interpreted when it is about to be executed. Therefore, functions represented using a script language will always need more time to get executed as compared with precompiled modules.

As overall performance is very important in the context of web visualisation, the proposed *FShape* node is VRML browser plug-in dependent, in which the VRML extension is implemented via customizing a VRML browser plug-in. The proposed function-based geometric node could be extended via different browser plug-ins, e.g. Blaxxun Contact 3D[17], OpenVRML[18], FreeWRL[19], and other customisable browsers.

In order to prove the concept proposed in this paper, we have extended Blaxxun Contact 3D to support *FShape* node. With the extension, viewers are able to visualise function-based objects in VRML paradigm. The extension of other VRML browsers can be done in a similar way, and we are working on it.

3.1. FShape node definition

The proposed *FShape* node definition is listed in Figure 2.

```
FShape {
ExposedField SFString sourceString []
ExposedField SFString sourceURL    []
exposedField
SFString objectName         "my_model"
field SFString sourceType       []
field SFBool ccw                TRUE
field SFBool  convex            TRUE
field SFFloat creaseAngle       0
field SFBool  normalPerVertex   TRUE
field SFBool  solid             TRUE
field SFBool  reduce            FALSE
field SFBool  searchVertex      FALSE
field SFFloat searchPer         0.1
field MFFloat modelPar          0.0
field SFInt32 gridSize          30
field SFInt32 boundingBoxMax    10
   }
```

Figure 2: *FShape node definition with default values.*

`sourceString` contains the definition of a function-defined object. This field has a higher priority than *sourceURL*, i.e. if the field is defined, it will be used as the object textual definition source, instead of using the URL specified in the *sourceURL* field.

`sourceURL` defines the actual path of the function-defined object source file. It can be set to either a URL or a physical file path, in which the function-based content could be retrieved accordingly.

`objectName` specifies the model name for a function-defined object. The default value for this field is "*my_model*".

`sourceType` identifies the type of the function-defined model referred in the scene. This field is prepared for the future extension to different types of function languages. The value of this field is used to associate the respective parser and polygoniser components with the main VRML plug-in module.

`ccw` specifies whether the points of a face are presented in a counter-clockwise (TRUE), clockwise or unknown order (FALSE). A face has two sides, and it is important to know which is the front side, and which is the backside. The *ccw* will determine which surface side is facing the observer.

`convex` specifies if the faces, being defined in an internal list of coordinate indexes defining the faces to be drawn, are convex or not. VRML can only draw convex faces. When concave faces are presented, the VRML browser will split the faces into smaller convex faces. This is a time consuming task. If this field is set to TRUE, the browser will not split the faces.

`creaseAngle` specifies an angle threshold. If two adjacent faces make an angle bigger than the *creaseAngle,* then the observer can see clearly where the two faces meet, the edge linking the two faces is sharp. Otherwise the edge linking the two faces will be smooth.

`normalPerVertex` specifies whether the normal of each vertex is computed or not.

`solid` determines whether the browser should draw both sides of a face or just the front side. If solid is TRUE, then the faces in *FShape* node form a solid shape. In this case there is no need to draw the backsides of each face. If solid is FALSE, then the browser will draw both sides for each face.

`reduce` determines whether the polygonisation process will produce the least number of polygons. The default value for this field is set to FALSE.

`searchVertex` turns on recursive searching for vertex positions. If this option is not invoked, then BiLinear Interpolation is used to determine a vertex position for non-snapped vertices.

`searchPer` specifies the search accuracy in percentage of cell lengths. For non-snapped vertices, the search is done along the edge. Otherwise, the search is performed around the normal. The values must lie within the range of 0.0001 to 10.0.

`modelPar` specifies model-specific parameters. For example, for the implicitly defined shapes, it may specify the isosurface value.

`gridSize` specifies the precision of rendering used for polygonisation. Its value depends on the polygoniser.

`boundingBoxMax` specifies the bounding box for the object being polygonised. Its +/- values are used to define the bounding box about each axis.

3.2. Interaction between modules in the visualisation pipeline

The visualisation pipeline in *FShape* node is categorized into three main modules, i.e. Seeker, Parser and Polygoniser components. Figure 3 presents the interaction between modules in the visualisation pipeline.

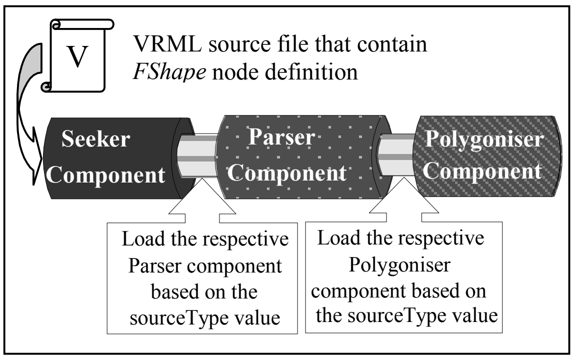
V
VRML source file that contain *FShape* node definition
Seeker Component
Parser Component
Polygoniser Component
Load the respective Parser component based on the sourceType value
Load the respective Polygoniser component based on the sourceType value

Figure 3: *Visualisation pipeline in FShape node.*

3.2.1. Seeker component

The Seeker component is used to obtain any type of function definition, which is either located inline with the VRML source or is remotely located at the content server. The inline source will always have higher priority than the external source, i.e. the inline source will always be used to describe the function-based model.

The Seeker component will be triggered first whenever the scene graph traversal engine encounters the *FShape* node. This component is activated to fetch the function model description or any data needed by the Parser component. The *FShape* core module will pass *sourceURL* and *sourceString* to the Seeker component. The component will return the respective data source to the core module. As data seeking is independent from any function-based shape modelling approach, the Seeker component is embedded as part of the core module.

3.2.2. Parser component

In order to allow for future extension of Parser component, the component is dynamically loaded into the *FShape* core module. The component is dependent on the data used to describe the model. For every supported function based model, the respective parser module is to be developed separately with a set of standard function calls.

The Parser component is created as a Win32 dynamic linked library (DLL) which has an extension "*.par*". The component is named according to its function-based shape modelling approach. This component must be stored into the relative *Parser* directory of the core module.

In order to locate the respective Parser component during runtime, the *sourceType*, that contains the component name, is used to load the respective module. The *FShape* core module will then create the respective instance for the Parser component. The required function for the Parser component is *Parse(string szSrc),* in which the data obtained by the Seeker component will be interpreted accordingly. The return result of this function is a class object called *CInterpreted*, as shown in Figure 4. During polygonisation, the *Calc()* will be heavily used by the Polygoniser component to compute the function values.

```
CInterpreted
{
 . . .
virtual void __stdcall parse
         (const string& model);
double calc
         (const vector<double>& X,
          const vector<double>& A);
double calc (const vector<double>&X);
 . . .// Some other member variables
}
```

Figure 4: *Class definition for CInterpreted.*

The data parsing is carried out in the memory, thus the performance is fast and efficient. However, it must be admitted that the memory usage for a very complex geometric model can be large.

3.2.3. Polygoniser component

Like the Parser component, the Polygoniser component is dependent on the chosen function language used to describe the model. For every supported function-based model, the respective Polygoniser component is to be developed and a set of standard function calls must be supported.

The Polygoniser component is created as a Win32 DLL which has an extension "*.pol*". The component is named according to its function-based shape modelling approach. This component must be stored into the relative *Polygoniser* directory of the core module.

The *FShape* core module uses the *sourceType* that contains the component name to locate the respective Polygoniser component during runtime. The *FShape* core module will then create the respective instance for the Polygoniser component. The *Init(...)* function must be implemented in Polygoniser component to initialise member values with *ccw, convex, creaseAngle, normalPerVertex, solid, reduce, searchVertex, searchPer, modelPar, gridSize, boundingBoxMax* and *CInterpreted* class object. Another required function for the Polygoniser component is *Calc(),* in which the *CInterpreted* object obtained from the Parser component will be used accordingly. The *CInterpreted* object, as shown in Figure 4, contains necessary values and functions in order to execute the polygonisation,

If the *Calc()* of the Polygoniser component is successful, the *FShape* core module will retrieve the point arrays like *coordVertex, normalVertex, coorIndex* and *normalIndex*. These arrays are then used to construct the polygonal facets representing the function-based models. Any geometric model with this information can be rendered directly.

4. Implementation of FShape node for F-Rep and its description language HyperFun

To illustrate the proposed design and to prove the correctness of our assumptions, we have implemented the *FShape* node for so-called F-Rep [20,21] representation with its dedicated high-level language *HyperFun*[22].

4.1. F-Rep and its applications

The F-Rep assumes that geometric shapes are represented with an inequality $f(x,y,z) \geq 0$, where the real function f is positive for the points inside the shape, equal to zero on its border and negative outside it. The function can be defined analytically, or with a function evaluation algorithm, or with tabulated values (e.g., CT or MRI volume data) and an appropriate interpolation procedure. Different operations can then be applied to the F-Rep geometric models to create new complex models. These operations include but not limited to affine, perspective and projective transformations, set-theoretic operations (union, intersection, subtraction), blending and morphing operations. They are defined in the function form as function superpositions. The result of any operation will be a function-defined shape, which can be used as an argument for other operations.

Various applications have been built using the F-Rep principles. For example, an application involving computer art is proposed by Sourin[23], in which the virtual embossing, wood-cutting and carving are done using the F-Reps of a shape, tools and interactive operations with them. The F-Rep principles have been also applied to obtain 3D textures on the constructive solids[24]. Other F-Rep applications include reconstruction from surface points and contours[25], simulation of NC machining[26], and the time dependent animation module[27].

In view of the power to describe complex multidimensional geometric models, it will be beneficial to the development of the web visualisation if the F-Rep shape modelling technique is adopted into the current web visualisation technologies. The compactness of the F-Rep defined objects and availability of a high-level modelling language for this representation have made F-Rep an attractive candidate to define objects to be transferred over the web.

4.2. HyperFun

HyperFun is a high-level modelling language specially designed to describe F-Rep models. The language is kept as simple as possible to allow the non-technically trained publishers mastering the tool with least effort. The language, while being simple, provides enough functions in creating quite complex geometric models. It supports all main notions in F-Rep, particularly those involving geometric objects and geometric operations.

As fundamental set-theoretic operations are important in F-Rep modelling, *HyperFun* has a series of special built-in operators with reserved symbols ("|" - union, "&" -intersection, "\" - subtraction, "~" - negation, "@" – Cartesian product). *HyperFun* also contains the system F-Rep library that can be used to represent geometric primitives and transformations. The most common primitives supported by *HyperFun* are 'Sphere', 'Torus', 'Ellipsoid', 'Cylinder', 'Blobby object', 'Metaball object', etc. Besides that, transformations such as blending, union/intersection, rotation, scaling and twisting, are included too. Functional expressions can also include references to previously defined geometric objects. The publisher can create his/her own library of objects for future use.

4.3. Integration with FShape node

The integration with *FShape* node can be achieved easily if the respective parser and polygoniser modules are available. For the F-Rep function models, the *HyperFun* developers kindly provided us their source codes, which helped us with developing the Parser and the Polygoniser modules since only modifications to fit those modules with basic interfaces defined by *FShape* node were required. The reader may download for testing the developed *FShape* plug-in from our project site at[28].

A sample VRML and *HyperFun* definitions are given in Figure 5 and 6 for the resulting VRML shape shown in Figure 7 and described in Figure 8. The resulting shape '*blend*' is defined by intersecting with blending the superellipsoid '*cube*' with the complex CSG shape named '*inside*', which is in turn created by unifying a sphere and two cylinders '*cylx*', '*cylz*' into a shape named '*spcyl*', followed by subtracting from it another cylinder '*hole*'.

In this example, the *HyperFun* definition source is referred via a file reference. Alternatively, the model definition can be embedded directly into the VRML code via *sourceString*.

```
#VRML V2.0 utf8
EXTERNPROTO FShape [...
]...

DEF my_model Transform {

   geometry DEF m FShape {
     sourceURL
     "http://www.model.com/md.hf"
     objectName       "my_Model"
   }

}}
```

Figure 5: *Using a file reference to specify a function based model description.*

```
my_model(x[3], a[1]){
-- HyperFun "Infinity" model by
-- Hidekazu YYoshida
array center[3], p[3], vertex[3];
center = [0, 0, 0];
p[1] = x[1]/10;
p[2] = x[2]/10;
p[3] = x[3]/10;
dX = p[1]^2;
dY = p[2]^2;
dZ = p[3]^2;
cylz =hfCylinderZ(p,center,0.548);
cylx = hfCylinderX(p,center,0.316);
cyl = cylz | cylx;
spcyl = fSphere(p,center,0.894)|cyl;
hole = hfCylinderY(p,center,0.316);
inside = spcyl \ hole;
cube = 0.9-dX*dX-dY*dY-dZ*dZ;
blend =
hfBlendInt(cube,inside,0.8,0.2,0.3);
my_model = blend;
}
```

Figure 6: *A sample HyperFun definition source.*

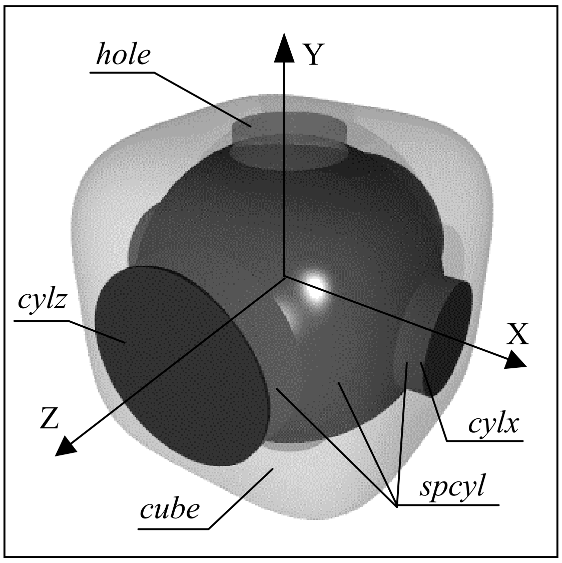

Figure 8: *A diagram of making the shape from Figure 7.*

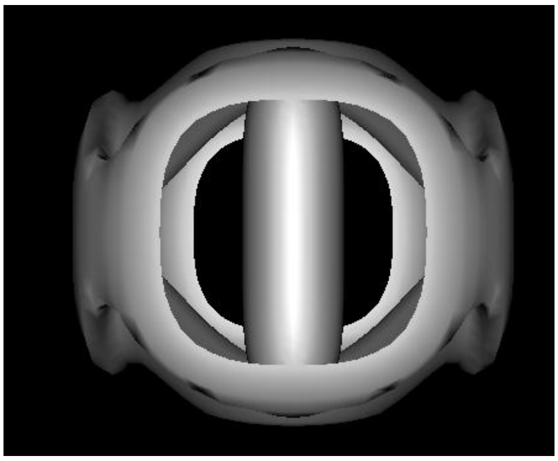

Figure 7: *A single complex model with a smooth surface.*

4.4. Performance improvement

Under different configurations shown in Table 1, the loading times for multiple complex objects such as the ones presented in Figure 9 are shown in Table 2. The grid density is set to 30 for all the selected models. The low density is chosen so that the rendering time is still acceptable for low-end machine, such as the Configuration C. The complexity of these models has caused the VRML file size to increase tremendously if those objects are represented using *IndexedFaceSet* node. Figure 10 illustrates the complexity of the shapes in a wire-frame view.

The loading time for the multiple objects represented using *FShape* nodes is shorter than the loading time for the multiple objects represented using *IndexedFaceSet* nodes. The time difference between these two representations for all the configurations becomes greater as compared to the time difference between them for loading a single object. This means that the efficiency of using *FShape* node, as compared to using *IndexedFaceSet* node, becomes more obvious for more complex objects. Objects represented using *FShape* nodes take longer time to load for slower machine. This may be due to the complex computation routines, which require faster CPU to process the *HyperFun* scripts and execute the polygonisation.

Configuration	**A**	**B**	**C**
Clock Speed	Athlon 650 MHz	P III 733 MHz	P II 300 MHz
RAM	128 M	128 M	96 M
Internet Connection	56 KB/s	100 MB/s	100 MB/s

Table 1: *Configurations tested.*

For Multiple Objects	Representation	
	IndexedFaceSet	FShape
File Size (Bytes)	3056690	3052
Loading Time for Config. A	272 sec	17 sec
Loading Time for Config. B	163 sec	15 sec
Loading Time for Config. C	188 sec	28 sec

Table 2: *Loading times for multiple objects.*

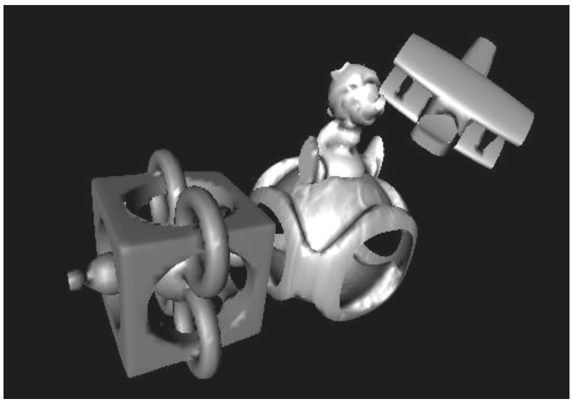

Figure 9: *Geometric models with smooth surfaces.*

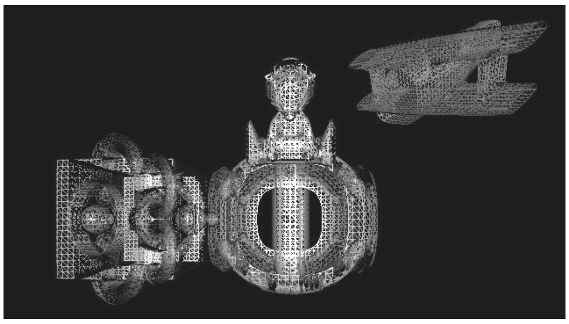

Figure 10: *Wire-frame geometric models.*

FShape node can coexist with other conventional VRML nodes, as shown in Figure 11. This indicates that the complex geometric models in a 3D world can be represented using *FShape* node, when creating other simple objects with VRML primitive nodes. The VRML source definition that consists of a *FShape* node and a *Sphere* node is listed in Figure 12.

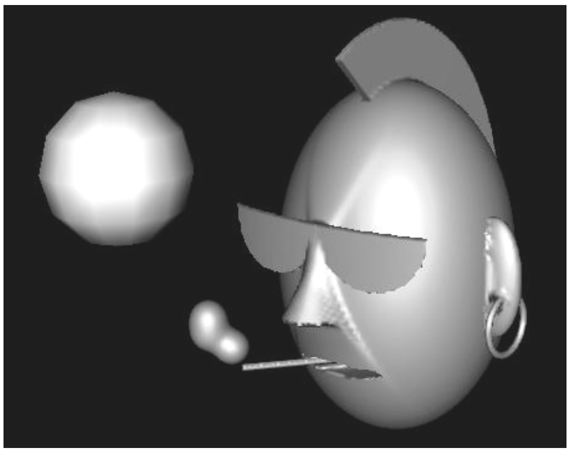

Figure 11: *FShape node and a VRML Sphere node.*

```
  . . .
DEF my_model Transform {
  children [
    Shape {
      appearance Appearance {
         . . .
        }
      geometry DEF mdl-FACES
FShape {
      sourceURL
      "http://www.mdl.com/m1.hf"
      ObjectName      "my_Model"
      }
    }
  ]
}

DEF my_model Transform {
  children [
    Shape {
      appearance Appearance {
         . . .
        }
      geometry DEF sp Sphere {
         radius 2.1
     }
        }
      ]
     }
```

Figure 12: *VRML file for FShape and Sphere nodes.*

5. Support for any function-defined models

Some function-defined models may not have a specific description language like *HyperFun*. They may be just represented by some mathematical functions, which are used with the supplied data values. In order to support any proprietary function-defined models in *FShape* node, the publishers have to develop their own respective Parser and Polygoniser components based on the predefined plug-in interface. Developing a Parser depends on the model and/or descriptive language used, while developing a Polygoniser depends only on the model. Therefore a set of polygonisers for the commonly used models can be developed and made available for the publishers. To illustrate it, we have applied the polygoniser used for visualising the F-Rep models in the project Interactive Function-based Shape Modeling[29]. The respective Parser module is a rather simple procedure in that case. It just reads the function model from the data file created interactively and makes it available for the Polygoniser. In Figure 13, a VRML scene with the function-defined model of a 3D carved crystal vase is displayed.

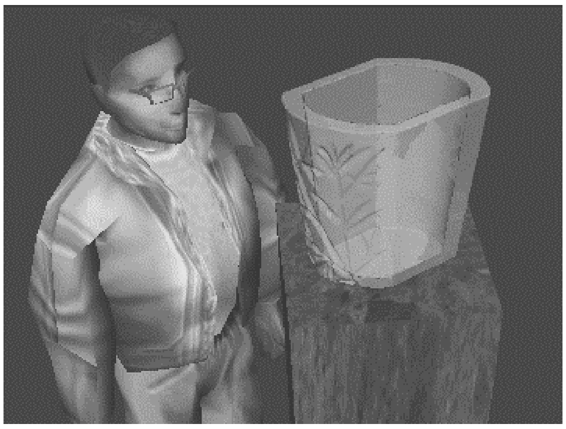

Figure 13: *A function-defined crystal vase.*

6. Conclusion and future work

In this article, a generic function-defined geometric shape node has been designed for VRML. The integration between function-defined models and VRML is proposed to implement through a VRML browser plug-in, where the customized node can be referred to as like a normal VRML node together with other conventional VRML nodes. Currently, the Blaxxun's Contact3D VRML plug-in has been extended to support this integration. The extension of other VRML browsers is on the way. The readers may download the developed *FShape* plug-in from our project site[28].

The use of the function-based geometric representations in VRML has opened new prospects for VRML modelling and indeed improved the overall performance under the bottleneck of the Internet bandwidth. Complex geometric models can be easily represented with a smaller file size, as compared to the conventional polygonal based representation.

Future work is planned in several directions. The inclusion of other function-defined models besides F-Rep and its dedicated language *HyperFun* will be done. Also, the standard function calls for the Parser and Polygoniser modules will be further improved to minimise future integration effort whenever a new function language is added into the system.

Acknowledgements

The authors thank the HyperFun developers and personally A.Pasko who kindly provided their codes and participated in the testing of the plug-in. Also, we thank K.Levinski for providing the polygoniser for the proprietary F-Rep models. Without these contributions, the prototype development would have definitely taken longer time.

References

1. K.Engel, R. Grosso, and T. Ertl. Progressive Isosurfaces on the Web. *Proc. Visualisation 98*, pages 37-40, 1998.

2. K.Engel, R. Westermann and T. Ertl. Isosurface Extraction Techniques for Web-based Volume Visualisation. *Proc. IEEE Visualisation '99*, pages 139–146, 1999.

3. H-P. Seidel, H. P. A. Lensch, M. Goesele and J. Kautz. A Framework for the Acquisition, Processing, Transmission,and Interactive Display of High Quality 3D Models on the Web. *Techical Report MPI-I-2001-4-002*, *Max-Planck-Institut f ̈ur Informatik, Germany. http://www.c-lab.de/ web3d2001/Workshops/ tutorial_HighQuality3DModels_web3d2001.pdf.*

4. E. Fogel, D. Cohen-Or, R. Ironi and T. Zvi. A Web Architecture for Progressive Delivery of 3D Content. *Proc. On 3D technologies for the World Wide Web (In Virtual Reality Modeling Language Symposium)*, pages 15–22, 2001.

5. G. Taubin, W. Horn, F. Lazarus, and J. Rossignac. Geometry coding and VRML. *Proc. of the IEEE*, 96:6, pages 1228-1243, 1998.

6. G. Taubin, A. Gueziec, W. Horn and F. Lazarus. Progressive Forest Split Compression. *Proc. of SIGGRAPH 98*, pages 123-132, 1998.

7. R. Pajarola and J. Rossignac. Compressed Progressive Meshes. *IEEE Transactions on Visualisation and Computer Graphics*, 6:1, pages 79–93, 2000.

8. Web3D specifications. http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm.

9. M. Alexa, J. Behr and W. Müller. The Morph Node. *Proc. of the Web3D-VRML 2000 Fifth Symposium on Virtual Reality Modeling Language*, pages 29–34, 2000.

10. GeoVRML 1.1 Specification. *http://www.geovrml.org.*

11. B. Wyvill and A. Guy. The Blob Tree, Implicit Modeling and VRML. *Proc. International Conference From the Desktop to the Webtop: Virtual Environments on the Internet*, *WWW and Networks, NMPFT, Bradford*, pages 193–206, 1997.

12. J. J. Pittet, A. Engel and B. Heymann. Visualizing 3D Data Obtained from Microscopy on the Internet. *JSB 125*, pages 123–132, 1999.

13. W. E. Lorensen and H. E. Cline. Marching Cubes: a High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21:4, pages 163-169,1987.

14. H. Grahn, T. Volk and H. J. Wolters. NURBS in VRML. *Proc. of the Web3D-VRML 2000 fifth symposium on Virtual Reality Modeling Language*, pages 35–43, 2000.

15. R. Ginis and D. Nadeau. Creating VRML Extensions to Support Scientific Visualisation. *Proc. of the 1995 Symposium on Virtual Reality Modeling Language*, pages 13–20, 1995.

16. C. Marrin, Proposal for a VRML 2.0 Informative Annex: External Authoring Interface Reference, *http://www.web3d.org/WorkingGroups/vrml-eai/history/eai_draft.html.*

17. Blaxxun Contact. *http://www.blaxxun.com.*

18. OpenVRML. *http://www.openvrml.org*.

19. FreeWRL. *http://freewrl.sourceforge.net*.

20. Shape Modeling and Computer Graphics with Real Functions. *http://wwwcis.k.hosei.ac.jp/~F-rep/.*

21. A. A. Pasko, V. D. Adzhiev, A. I. Sourin, V. V. Savchenko. Function Representation in Geometric Modeling: Concepts, Implementations and Applications. *The Visual Computer*, vol.11, No.8, pages 429-446, 1995.

22. A. Pasko, V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov and V. Savchenko. HyperFun Project: a Framework for Collaborative Multidimensional F-rep Modeling. *Eurographics/ACM SIGGRAPH Workshop Implicit Surfaces '99*, pages 59–69, 1999.

23. A. Sourin. Functionally Based Virtual Computer Art. *Proc. The 2001 ACM Symposium on Interactive Computer Graphics, I3D2001*, pages 77-84, 2001.

24. A. A. Pasko and V. V. Savchenko. Solid Noise in the Constructive Solid Geometry. *Proc. EDUGRAPHICS '93 and Computer Graphics '93*, ACM, Portugal, pages 351–357, 1993.

25. V. V. Savchenko and A. A. Pasko. Reconstruction from Contour Data and Sculpting 3D Objects. *Proc. of Second International Symposium on Computer Aided Surgery ISCAS'95*, pages 56-57, 1995.

26. A. I. Sourin and A. A. Pasko. Function Representation for Sweeping by A Moving Solid. *IEEE Transactions on Visualisation and Computer Graphics*, 2:1, Special issue on solid modeling, pages 11-18, 1996.

27. E. Fausett, A. Pasko and V. Adzhiev. Space-Time and Higher Dimensional Modeling for Animation. *Proc. of the Computer Animation 2000 (CA'00).* pages 140-145, 2000.

28. Function-based Web Visualisation project site *http://www.ntu.edu.sg/home/assourin/FVRML.htm.*

29. K. Levinski and A.Sourin, Interactive Polygonisation for Function-based Shape Modelling, *Eurographics 2002, short presentation*.