Web Visualization of the Function-defined Virtual Worlds

Lai Feng Min¹ Alexei Sourin² Konstantin Levinski³
¹Creative Technology Ltd., Singapore ²Nanyang Technological University, Singapore
³Moscow Institute of Physics and Technology, Russia
fmlai@ctl.creative.com assourin@ntu.edu.sg kostya_sing@hotmail.com

Abstract

This paper describes how the web visualization can be greatly improved using the function-based shape modeling technique. The improvement is possible because the proposed function-defined VRML shape node allows the content creators to describe any complex models with relatively small functions compared to the large-size polygonal mesh based VRML nodes. These function-defined shapes can be used together with the common VRML shapes. The proposed node has a few implementations capable of visualizing geometric shapes defined with HyperFun language as well as in any proprietary function-defined data format. For fast visualization of the function-defined shapes, we have developed an improved continuation polygonization algorithm specifically designed for VRML visualization. The design, the implementation details, and the application examples of the proposed node are discussed.

1. Web visualization

Web visualization 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 visualization 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 visualization. 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 bidirectional 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 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 Modeling Language (VRML) [8] uses this representation for complex objects.

Although there are many workarounds to improve the overall experience in 3D web visualization, the mainstream of the current web visualization 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 modeling file size is kept within an optimized range. From the different web visualization models or techniques available today, some techniques are adopted to improve the performance of visualization; while some are used to transmit the visualization data interactively based on the level of details and the network load. However, the root of the problem is the actual representation of the visualization data to be transmitted over the network. The compactness of the visual representation directly affects the overall user experience in interactive web visualization.

In this project, we design and implement a generic function-based shape node for VRML. In Section 2, the function-based approach to shape modeling in web visualization 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-based shape node developed for the particular function model F-Rep with its dedicated description language HyperFun. In Section 5, we illustrate how a proprietary function model can be plugged-into our architecture. In Section 6, the performance analysis is given. In the last section, further issues are discussed and conclusions are made.

2. Function-based shape modeling over the Internet

2.1. Function-based shape modeling

Function-based shape modeling is becoming increasingly popular in computer graphics. The idea of function-based approach to shape modeling 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 polygonization followed by fast polygon rendering is used. Alternatively, the function-defined shapes can be voxelized 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, revolutionize the ways of shape representation. In the next subsections, we select the direction of expanding function-based modeling to web visualization, overview the existing projects attempting such visualization, and propose our own solution to this problem.

2.2. VRML and its existing extensions

VRML is the most common format adopted to define 3D objects in the virtual web spaces. We have chosen it to integrate with function-based modeling technique for web visualization. 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. This 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 visualization of complex CAD models in VRML. Meanwhile, Ginis and Nadeau [15] have presented new VRML extensions for scientific visualizations. 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 visualization.

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 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 polygonization 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 polygonization has completed, the complex geometric model will be presented to the client.

With the proposed architecture design, a new function-based geometric node, called FShape, has been proposed for VRML. The FShape node is designed in such a way that it offers a generic infrastructure to support various function-defined models to be plugged-in during runtime. This can be achieved if the respective parsing and polygonization modules are implemented during runtime.
publishers can present any proprietary function-defined model to the viewers since they can develop the respective parsing and polygonization modules.

3. FShape node implementation

A VRML node extension can be achieved either via plug-in customization or through VRML EAI interface [16]. For plug-in customization, 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 customization 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 could be 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 a serious performance setback. The polygonization 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 visualization, 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 customizable 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 visualize 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 1.

\[
\text{FShape} \\
\text{ExposedField SFString sourceString} \\
\text{ExposedField SFString sourceURL} \\
\text{exposedField SFString objectName} = "my_model" \\
\text{field SFString sourceType} [] \\
\text{field SFBool ccw} = \text{TRUE} \\
\text{field SFBool convex} = \text{TRUE} \\
\text{field SFFloat creaseAngle} = 0 \\
\text{field SFBool normalPerVertex} = \text{TRUE} \\
\text{field SFBool solid} = \text{TRUE} \\
\text{field SFBool reduce} = \text{FALSE} \\
\text{field SFBool searchVertex} = \text{FALSE} \\
\text{field SFFloat searchPer} = 0.1 \\
\text{field MFFloat modelPar} = 0.0 \\
\text{field MFInt32 gridSize} = 30 \\
\text{field MFFloat boundingBoxMax} = 10 \\
\]

\[\text{Figure 1. FShape node definition with default values}\]

**sourceString** specifies the definition of 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 polygonizer 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).

**convex** specifies if the faces, being defined in an internal list of coordinate indexes defining the faces to be drawn, are convex or not.

**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.

**reduce** determines whether the polygonization 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.

\[\text{sourceURL} \text{ defines the actual path of the function-defined object source file. It can be set to either a URL or a physical file path, in which the function-based content could be retrieved accordingly.}\]
**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 specifies the isosurface value.

**gridSize** specifies the grid density used for polygonization. The value depends on the polygonizer used.

**boundingBoxMax** specifies the bounding box for the object being polygonised. When one argument is given it's +/- values are used to define the bounding box about each axis. Two arguments give the minimum and maximum values around each axis. Three arguments specify the +/- values of the X, Y, and Z axis respectively.

### 3.2. Interaction between modules in the visualization pipeline

The visualization pipeline in *FShape* node is categorized into three main modules: **Seeker, Parser** and **Polygonizer** components.

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 precedence 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 modeling approach, the Seeker component is embedded as part of the core module.

In order to allow for future extension of **Parser** component in runtime, 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 “.pol”. The component is named according to its function-based shape modeling approach. This component must be stored into the relative **Polygonizer** directory of the core module. The *FShape* core module uses the **sourceType** that contains the component name to locate the respective Polygonizer component during runtime. 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 2. During polygonization, the **Calc()** will be heavily used by the Polygonizer component to compute the function values. 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.

```cpp
CInterpreted
{
    void 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 2. Class definition for CInterpreted**

Like the Parser component, the **Polygonizer** component is dependent on the chosen function language used to describe the model. For every supported function-based model, the respective Polygonizer component is to be developed and a set of standard function calls must be supported. The Polygonizer component is created as a Win32 DLL which has an extension “.pol”. The component is named according to its function-based shape modeling approach. This component must be stored into the relative **Polygonizer** directory of the core module. The *FShape* core module uses the **sourceType** that contains the component name to locate the respective Polygonizer component during runtime. The *FShape* core module will then create the respective instance for the Polygonizer component. The **Init(...)** function must be implemented in Polygonizer 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 Polygonizer component is **Calc()**, in which the **CInterpreted** object obtained from the Parser component will be used accordingly. The **CInterpreted** object, as shown in Figure 2, contains necessary values and functions in order to execute the polygonization. If the **Calc()** of the Polygonizer 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
4. Support for F-Rep model and its description language HyperFun

To illustrate the proposed plug-in design and to prove the correctness of our assumptions, we have implemented the FShape node for the 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 visualization if the F-Rep shape modeling technique is adopted into the current web visualization technologies. The compactness of the F-Rep defined objects and availability of a high-level modeling 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 modeling 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 modeling, 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 Polygonizer modules are available. For the F-Rep function models, the HyperFun developers have kindly provided the original Parser and Polygonizer modules. Only modifications to fit those modules with basic interfaces defined by FShape node are needed. The reader may download for testing the developed FShape plug-in from our project site at [28].

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

DEF my_model Transform {
    geometry DEF m FShape {
        sourceURL "http://www.model.com/md.hf"
        sourceType "hyperfun"
    }
}
```

Figure 3. Using a file reference to specify a function based model description

A sample VRML and HyperFun definitions are given in Figure 3 and 4 for the resulting VRML shape shown in Figure 5. 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.

```plaintext
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 4. A sample HyperFun definition source

Figure 5. A single complex model with a smooth surface.

5. Support for any proprietary 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 Polygonizer components based on the predefined plug-in interfaces. Developing a Parser depends on the model and/or language used, while developing a Polygonizer depends only on the model. Therefore a set of polygonizers for the commonly used models can be developed and made available for the publishers. For obtaining a polygonal representation of a function-defined shape, a significant amount of points on or near its surface is to be calculated. This task involves multiple function evaluations, which can be time consuming. For further VRML rendering, we usually need only the polygons on the surface of the shape. Also, these polygons are to be calculated as fast as possible. For parametric function representations, the polygonization is rather a straightforward task, while it is not so obvious for implicit functions. Taking it into account, we developed the Polygonizer, which works efficiently for the implicitly defined and F-Rep defined shapes. This algorithm, described in details in [29], is an extension and improvement of the so-called continuation algorithm, which defines only polygons on the surface of the shape.

The developed Polygonizer is used for visualizing the F-Rep models from the project Interactive Function-based Shape Modeling [28]. 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 Polygonizer. By identifying the function model as myscene, the Parser and Polygonizer components are separately named as myscene.par and myscene.pol. These two components are then copied to the directories as shown in Figure 6, where the parent directory, C:\Blaxxun contains the main VRML plug-in.

Figure 6. Locations to install the parser and Polygonizer

The FShape node will dynamically load the corresponding Parser and Polygonizer components based on the string value stored in sourceType field. The VRML source code that links to this proprietary function model is shown in Figure 7 and the examples of using such function-defined shapes in a VRML scene are presented in Figure 8.

Since the Polygonizer was developed for a large class of models, for using any other implicitly or F-Rep defined models, only the respective Parser procedure is to be written. It just should be able to evaluate the shape defining function f(x,y,z) for any given coordinates.

FShape node can coexist with other conventional VRML nodes, as shown in Figure 8. Therefore, the complex geometric models in a 3D world can be
represented using \textit{FShape} node, while creating other simple objects with VRML primitive nodes.

\begin{verbatim}
# VRML V2.0 utf8
EXTERNPROTO FShape [...]...

DEF my_model Transform {
    geometry DEF m FShape {
        sourceURL "http://www.model.com/any.dat"
        sourceType        "myscene"
        gridSize          16
    }
}
\end{verbatim}

\textbf{Figure 7.} VRML source for the proprietary function model

\begin{verbatim}
DEF my_model Transform {
    geometry DEF m FShape {
        sourceURL "http://www.model.com/any.dat"
        sourceType        "myscene"
        gridSize          16
    }
}
\end{verbatim}

\textbf{Figure 8.} Function-defined wood-carving and crystal vase

\textbf{Figure 9.} Geometric models with smooth surfaces

\section*{6. Performance analysis}

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. A low polygonal mesh quality was chosen to make the rendering time acceptable for low-end machine, such as the Configuration C. The complexity of these models would cause the VRML file size to increase tremendously if those objects have been represented using \textit{IndexedFaceSet} node.

It has appeared that the loading time for the multiple objects represented using \textit{FShape} nodes is shorter than the loading time for the multiple objects represented using \textit{IndexedFaceSet} nodes. The loading time includes the downloading time and the rendering time. 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 \textit{FShape} node, as compared to using \textit{IndexedFaceSet} node, becomes more obvious for more complex objects. Objects represented using \textit{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 \textit{HyperFun} scripts and execute the polygonization.

\begin{table}[h]
\centering
\caption{Configurations tested}
\begin{tabular}{|c|c|c|c|}
\hline
Configuration & A & B & C \\
\hline
Speed (MHz) & Athlon 650 & P III 733 & P II 300 \\
\hline
RAM & 128 M & 128 M & 96 M \\
\hline
Connection & 56 KBS & 100 MBs & 100 MBs \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{Loading times for multiple objects}
\begin{tabular}{|c|c|c|c|}
\hline
For Multiple Objects & Representation & IndexedFaceSet & FShape \\
\hline
File Size (Bytes) & 3056690 & 3052 \\
\hline
Loading Time for A & 272 sec & 17 sec \\
\hline
Loading Time for B & 163 sec & 15 sec \\
\hline
Loading Time for C & 188 sec & 28 sec \\
\hline
\end{tabular}
\end{table}

\section*{7. Conclusion and future work}

In this article, a generic function-based geometric shape node has been defined 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 browser has been extended to support this integration. The extension of other VRML browsers is on the way.

The use of the function-based geometric representations in VRML has open new prospects for VRML modeling 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.

Currently, we are working on inclusion of other function-defined models besides F-Rep and its dedicated language HyperFun. The developed software is available for downloading.

Acknowledgements

This paper has become possible because of the RFBR grant 00-07-90165.

References