Interactive Modeling and Visualization of F-rep Solids with an Extendable User Interface

by

Yuichiro Goto

March 2002
The thesis titled

*Interactive Modeling and Visualization of F-rep Solids with an Extendable User Interface*

by

Yuichiro Goto

is reviewed and approved by:

---

Main referee

*Professor*  
Shunji Mori  
*Professor*  
Daming Wei  
*Assistant Professor*  
Jung-pil Shin

The University of Aizu

*March 2002*
Contents

Chapter 1 Introduction 1

Chapter 2 Interactive Modeling of Convolution Surfaces with an Extendable User Interface 2
  2.1 Convolution Surfaces 3
  2.2 HyperFun 4
  2.3 Modeler with an Extendable User Interface 4
    2.3.1 Implementation 4
    2.3.2 Features 5
    2.3.3 Binding primitives 6
    HFModel 6
    Parameters 7
    Defaults 8
    Visual 8
  2.4 Example 9

Chapter 3 Polygonization Based on Marching Cubes Algorithm without Ambiguities 12
  3.1 Overview 12
    3.1.1 Basic Idea of Polygonization 12
    3.1.2 Marching Cubes Algorithm 12
    3.1.3 Disadvantage of Marching Cubes Algorithm 13
  3.2 Fast Polygonization Method Based on Marching Cubes Algorithm 14
    3.2.1 Disambiguation 19
    3.2.2 Cell Polygonization Procedure 19
  3.3 Application 20
    3.3.1 Demos for the Magic Box “Paco” 20

Chapter 4 Internet-based Interactive Modeler for F-rep Solids 22
  4.1 Interactive Modeler 22
    4.1.1 Implementation 22

Chapter 5 Conclusions 25

References 26
## List of Figures

- **Figure 2.1** Examples of convolution surfaces .................................................. 4
- **Figure 2.2** Screenshot of the modeler ................................................................. 5
- **Figure 2.3** Polygonized surface ........................................................................... 6
- **Figure 2.4** Example of initialization file ............................................................... 7
- **Figure 2.5** Initialization file for the body of the fish ........................................... 9
- **Figure 2.6** The modeler with some triangles ......................................................... 10
- **Figure 2.7** Body of the fish ................................................................................. 11
- **Figure 2.8** Bones of the fish ............................................................................... 11
- **Figure 2.9** Fish image ......................................................................................... 11
- **Figure 2.10** Ray-traced fish image ..................................................................... 11
- **Figure 3.1** 15 basic configurations ..................................................................... 13
- **Figure 3.2** Ambiguous faces .............................................................................. 14
- **Figure 3.3** Structure of look-up table ................................................................. 15
- **Figure 3.4** Cell indexing convention ................................................................. 15
- **Figure 3.5** Configuration 3 .............................................................................. 15
- **Figure 3.6** Configuration 5 .............................................................................. 16
- **Figure 3.7** Two surface vertex connections for an ambiguous face .................. 17
- **Figure 3.8** Inspection order for cell faces ............................................................ 17
- **Figure 3.9** Configuration 26 ............................................................................. 18
- **Figure 3.10** Local linear and bilinear interpolation on the cell face .................... 19
- **Figure 3.11** Screenshot of the Magic Box “Paco” .............................................. 20
- **Figure 3.12** Interactive metamorphosis and twisting demo ............................... 21
- **Figure 4.1** Interactive HyperFun modeler applet .............................................. 23
- **Figure 4.2** Translation sequence from HyperFun program to triangle meshes 24
List of Tables

Table 2.1  Polygonization time .......................... 10
I wish to thank Dr. Shunji Mori for his support and encouragement. I wish to express my appreciation to Associate Professor Carl Vilbranct for his excellent advice and diligent efforts to guide me through this project.

I would like to thank Professor Alexander Pasko, Jody Vilbrandt and Kazuhiro Mochizuki for their help and support.
Abstract

This thesis presents interactive modeling and visualization techniques for F-rep solids with an extendable user interface. An extendable user interface offers flexibility for modeling and visualization. F-rep (Function representation) is a unifying approach to deal with different types of shape models, such as classic and skeleton based implicit surfaces, set-theoretic solids, sweeps, volumetric objects, parametric models, procedural models, and so on. To specify F-rep solids, HyperFun is used in this thesis. HyperFun is a high-level language for modeling F-rep solids. HyperFun can be used as an exchange protocol of F-rep models between different software tools, and can be used as lightweight protocol to exchange F-rep models through the Internet. First, interactive modeling of convolution surface with an extendable user interface is presented as a specific case of F-rep solid modeling. Then, to increase interactivity of modeling and visualization, an ambiguity-free polygonization method based on marching cubes algorithm is introduced. Finally, Internet-based interactive shape modeling is described.
Chapter 1

Introduction

In this thesis, we introduce interactive modeling and visualization techniques of implicit surfaces and more general F-rep solids with an extendable user interface.

Implicit surfaces are quite flexible geometric shapes, and are used in many applications, for example, computer graphics, computer aided design, and molecular modeling. An implicit surface is defined as the zero set of a rule which is described either mathematically or procedurally. Algebraic and skeleton-based implicit surfaces, convolution surfaces, and distance-based models are defined in this way. Set-theoretic operations are applicable to implicit surfaces. Complex objects can be constructed applying set-theoretic operations to simple primitives. Implicit surfaces can not only define various shapes but can be deformed in many ways, such as twisting, sweeping, tapering, and so on.

Function representation, F-rep for short, is a more general representation of a geometric object than implicit surfaces. F-rep defines a geometric object by a single real continuous function of several variables as $F(x_1, x_2, x_3, ..., x_n) \geq 0$. F-rep can act as exchange protocol of functionally-defined geometric objects.

This thesis is organized as follows. Chapter 2 explains an interactive modeling technique of convolution surfaces with an extendable user interface. An algorithm for fast polygonization of implicit surfaces using marching cubes algorithms and local bilinear interpolation of the corner function values and its applications are described in Chapter 3. Chapter 4 introduces internet-based client-server type modeling environment using HyperFun. Conclusions are discussed in Chapter 5.
Chapter 2

Interactive Modeling of Convolution Surfaces with an Extendable User Interface

In this chapter, interactive modeling of convolution surfaces with an extendable user interface is presented as a special case of interactive modeling of F-rep solids. Using an extendable user interface, the modeler is not bound by limited types of convolution surfaces. The user can introduce new primitives in the modeler without rewriting the source code of the modeler.

A convolution surface is a kind of implicit surface, which is defined using integration of a kernel function over a skeleton consisting of different geometric elements. Properties of convolution, such as superposition, make control of smooth implicit surfaces using skeletal elements quite intuitive. However, since introduction of them in 1991 by Bloomenthal and Shoemake [1], convolution surfaces were considered an elegant but not practical solution because of the problems with slow and not precise numerical integration. Analytical solutions for some kernel functions and skeletal elements were found by Sherstyuk [2, 3]. Analytically defined convolution surfaces can be polygonized in near real-time rate and can be used for interactive modeling. Since there are different types of kernel functions and skeletal elements, various types of convolution surfaces exist. For this reason, an extensibility of the modeler is very useful.

The modeler exports the created model to the HyperFun program [4, 6]. HyperFun is a simple high-level language intended for describing F-rep models [5]. F-rep defines a geometric object in the form of $F(x_1, x_2, x_3, ..., x_n) \geq 0$. F-rep models are more general than traditional implicit surfaces. A convolution surface can be thought of as a specific F-rep model. There are several advantages of using HyperFun as the output. HyperFun serves as an exchange protocol for F-rep models: the models described in HyperFun can be transferred between different tools. Since HyperFun can describe complex models with a small amount of code, it can be used as lightweight protocol between internet-based visualization software. Various modeling systems supporting HyperFun have been developed on several computer platforms such as Unix, GNU/Linux and Windows. Since HyperFun is designed to be very simple, the user can easily
2.1 Convolution Surfaces

An implicit surface is defined by a field function or a potential function $f$:

$$\{ \mathbf{p} \in \mathbb{R}^3 | f(\mathbf{p}) = c \}$$

where $c$ is the isopotential value of the surface. A convolution surface is a specific implicit surface. A field function of a convolution surface [2] is defined by a convolution of a skeletal element $s$ and a kernel function $h$:

$$f(\mathbf{p}) = \int_{\mathbb{R}^3} s(\mathbf{r}) h(\mathbf{p} - \mathbf{r}) d\mathbf{r}$$

A skeletal element defines a skeleton of an object. Points, lines, arcs, and triangles are used as skeletal elements. A kernel function defines a distribution of a potential value for each point of the skeleton element. As a kernel function, Cauchy, Gaussian, inverse, and inverse square functions are used for our primitives. The sophisticated features of convolution surfaces are that free-form soft objects can be easily generated, and skeleton-based design is allowed. Figure 2.1 shows examples of convolution surfaces.
2.2 HyperFun

HyperFun is a simple high-level language designed for specification of functionally-defined models (F-rep) in the form of \( F(x_1, x_2, x_3, ..., x_n) \geq 0 \). HyperFun is simple, but has enough facilities for creating complex F-rep models [4]. In HyperFun, an F-rep model can be constructed using assignments, conditional selections, and iterations as in traditional programming languages. Conventional arithmetic and relational operators, standard mathematical functions, built-in set-theoretic operators, and F-rep system libraries are available. The user can extend the F-rep library on different language levels (HyperFun, C, Java). Several types of convolution surface primitives are available in the F-rep library. Convolution surface primitives are an example of extensions of the F-rep library on the C level. The subject of this work is the interactive modeler extension without its reprogramming.

2.3 Modeler with an Extendable User Interface

2.3.1 Implementation

MAM/VRS [7] is used for implementing the modeler. MAM/VRS is a multi-platform C++ graphics library. The library uses OpenGL or Mesa for rendering and supports several toolkits for constructing GUIs. We choose Tcl/Tk [8] as a toolkit, because Tcl/Tk is free and available on Unix, GNU/Linux, and Windows.
2.3.2 Features

Figure 2.2 is the screenshot of the modeler. The modeler has four windows. The upper two windows and lower left window have orthogonal views. The lower right window has perspective view. The user interacts with the modeler using four windows, but an operation such as moving a vertex is not allowed in the perspective window.

The modeler is extendable by convolution surface primitives, which become built-in primitives of the modeler. If convolution surface primitives using new kernels are added to the F-rep library, it is possible to use those primitives without rewriting the source code of the modeler. The modeler is different from usual modelers in this aspect. Primitive bindings are specified in an initialization file. The modeler must be invoked with this file. Available built-in primitives of the modeler are points, lines, cylinders, and meshes. Cylinders can be used as offset surfaces that represent an approximation of polygonized convolution surfaces. Meshes mean indexed triangles, that is triangles are represented in a mesh by a vertex array and an index array.

![Screenshot of the modeler](image.png)

Figure 2.2: Screenshot of the modeler

Now, we describe an outline of modeling procedures using the modeler. First, a file for binding convolution surface primitives must be prepared. Next, the modeler is invoked with the file. If the file has no syntax error, the modeler’s interface appears on the screen. If default values are used, some objects may appear on the working areas of the modeler. Then, the user interacts with the modeler to create and manipulate the objects. During the interaction, the user can obtain a polygonized surface for the
current model. Figure 2.3 shows an example of a polygonized surface. Finally, the modeler generates the HyperFun program for the created model.

![Figure 2.3: Polygonized surface](image)

### 2.3.3 Binding primitives

This section describes the format of an initialization file for the primitives binding. This file is important to inform the modeler what F-rep primitives of convolution surfaces are used, the way to display them during interactive modeling, and how to generate HyperFun programs for them. Primitives are specified in a function form that is like \( h_f(name)(par_1, par_2, ..., par_k) \). The function form is the same as the one used in HyperFun. The binding is achieved by passing parameters of convolution surface primitives to built-in primitives of the modeler. If bindings are successful, the modeler can change the value of parameters. The parameters can have textual information and default values. Textual information explains the meaning of a parameter. The file consists of four blocks specifying bindings:

- HFModel
- Parameters
- Defaults
- Visual

These blocks can be placed in any order. In the file, comments are allowed after “—”. Let us explain each block with examples. Figure 2.4 is an example of the initialization file.

**HFModel**

A HyperFun model of the convolution surface primitive is specified in this block. Textual information of a primitive, type of the F-rep element, and F-rep library function for the primitive are declared. There are two types of F-rep elements, primitive and
operation, but now only “primitive” can be specified. An F-rep primitive is declared with its function used in HyperFun, that is \( hf \langle name \rangle \) is replaced with an identifier of a specific convolution surface primitive. The parameter “\( x \)” is reserved in HyperFun for the array of coordinate variables.

HFModel
{
   "mesh primitive"
   primitive
   hfConvMeshR(x, v, i, 1.5)
}

Parameters
{
   {"vector" array v[]}
   {"index" array i[]}
}

Visual
   "triangle skeleton"
   shape
   Mesh(v, i)
}

Figure 2.4: Example of initialization file

Parameters
Declarations of primitive parameters are presented in this block. Textual information of a parameter, type of a parameter, and parameter name are needed. The textual information explains the meaning of the parameter. This information is specified by a string. The types of parameters are “real” and “array.” The words “\( x \)” and “\( a \)” should not be declared, because such words are reserved in HyperFun and lead to
syntax error during the interpretation of generated programs by the modeler. Also, “If,” “while,” and other reserved words should not be used. The form “name[size]” is used, if a parameter is an array. Size of the parameter is an option: an empty size array is allowed, but it can not have default values.

Parameters
{
  
  "point coordinates"
  array
  center[3]

  
  "half-axes along x" real b
  "half-axes along y" real c
  "half-axes along z" real d

}{

Defaults
Each parameter can have a default value. Default values of parameters are given in this block. Values are assigned to parameters using assignment operator “=.” If a parameter is an array type, there are two ways to assign values. The first way is to assign values to each element. The second way is to assign all values at once. Assignment between parameters is not allowed.

Defaults
{
  
  -- assign values to each element --
  -- center[0] = 0.0
  -- center[1] = 0.0
  -- center[2] = 0.0

  -- assign all values at once --
  center = [0.0, 0.0, 0.0]
  b = 1.0
  c = 1.0
  d = 1.0

  -- b = center[0] is not allowed. --

}{

Visual
In this block, a built-in primitive of the modeler is specified as a visual representation of a convolution surface primitive. Same as in the HFModel block, textural information of a primitive, primitive type, and primitive in a function form are given. Now, the only available type is “shape.” Parameters of F-rep primitives are passed to built-in
primitives. Available built-in primitives are “Point,” “Line,” “Cylinder,” and “Mesh.” For each built-in primitive, we check types of parameters, array size, relations between arrays. For example, built-in Line primitive has two array parameters (start points and end points of each segment), each array size should be a multiple of 3, and arrays should have the same size.

Visual
{
    "Point Skeleton"
    shape
    Point(center)
}

HFModel {
    "mesh primitive"
    primitive
    hfConvMeshR(x, v, i, r)
}

Parameters {
    {"vector" array v[]}
    {"index" array i[]}
    {"radius" real r}
}

Defaults {
    r = 3.0
}

Visual {
    "triangle skeleton"
    shape
    Mesh(v, i)
}

Figure 2.5: Initialization file for the body of the fish

2.4 Example

In this section, an example is given using the modeler. Figure 2.10 is the final image of the example. This fish’s skeleton consists of lines and triangles. HyperFun function
“hfConvLineR” is used for lines, “hfConvMeshR” is used for triangles. Both primitives utilize Cauchy kernel [2, 3]. The image is ray-traced by the POV-Ray system. First of all, to deal with convolution surface primitives, the initialization file for binding primitives has to be prepared.

![Image of the modeler with triangles](image)

Figure 2.5 is an initialize file for the body of the fish. Only triangles are used for the body. Figure 2.6 is a screenshot of the modeler invoked with the file in Figure 2.5. Triangles are added on the working area of the modeler. To see a precise shape of a current model, polygonization is useful. Figure 2.7 is the result of polygonization of the model in Figure 2.6. Figure 2.8 shows the bones of the fish. To construct the bones, lines and triangles are used. The complete model of the fish is shown in Figure 2.9. The model contains 13 triangles and 10 lines. Polygonization is done on a SGI O2 workstation with 180 MHz MIPS R5000 processor. Table 2.1 shows polygonization time for a single line and triangle, and the fish model in Figure 2.9.

<table>
<thead>
<tr>
<th></th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>single line</td>
<td>4.970000</td>
</tr>
<tr>
<td>single triangle</td>
<td>6.430000</td>
</tr>
<tr>
<td>fish model</td>
<td>77.010000</td>
</tr>
</tbody>
</table>

Table 2.1: Polygonization time
Figure 2.7: Body of the fish

Figure 2.8: Bones of the fish

Figure 2.9: Fish image

Figure 2.10: Ray-traced fish image
Chapter 3

Polygonization Based on Marching Cubes Algorithm without Ambiguities

In this chapter, combining the marching cubes algorithm [10] and the disambiguation algorithm described in [11], a fast ambiguity-free polygonization method is introduced.

3.1 Overview

3.1.1 Basic Idea of Polygonization

Polygonization is the process for generating polygonal meshes from an implicit surface. The polygonization takes two phases: spatial partitioning and cell polygonization. There are different classes of the polygonization, the following schemes are used in this thesis, which is called exhaustive enumeration in [9]. Three-dimensional space is limited using a bounding box, \( x \in [x_1, x_2], y \in [y_1, y_2], z \in [z_1, z_2] \), and the bounding box is partitioned into cubic cells. The function representing the implicit surface is evaluated at cell corners. A cell edge in which one corner has the positive function value and the other has the negative function value is intersected by the implicit surface. The intersection point can be found using the linear interpolation of the function values sampled at end points of the edge. The intersection points are connected inside each cell to form a set of polygons.

3.1.2 Marching Cubes Algorithm

Marching cubes algorithm [10] is used at the cell polygonization phase. The configuration of the set of polygons for a cubic cell is determined by the number of cell corners in which the function value is positive. Since each cell corner can take the binary state, in which the function value is positive or negative, there are 256 possible configurations for 8 corners. The connectivities of cell edges for 256 possible configurations are stored in a look-up table. An eight bit number in which each bit is corresponding to a cell corner is used as an index for the look-up table. The 256 possible configurations are reduced to 15 basic configurations using rotation and inversion of the state
of 8 corners, see figure 3.1. A red cell corner contains negative function values. The intersection points are assumed to be located at the middle of cell edges.

Figure 3.1: 15 basic configurations

3.1.3 Disadvantage of Marching Cubes Algorithm

Marching cubes algorithm is very useful, but it does not consider ambiguities. Ambiguity is a problem which occurs in the polygonization using cubic cells. Figure 3.2 shows ambiguities occurring on cell faces. Two ways of intersection of a cell face and the implicit surface are considered for an ambiguous face and thus there are $2^n$
possible configurations for a cell if a cell contains \( n \) ambiguous faces. Since marching cubes algorithm chooses an arbitrary surface vertex connection for an ambiguous face, it does not deal with ambiguities consistently along shared faces and sometimes holes are found on the generated surface.

\[ \begin{array}{c}
\text{Figure 3.2: Ambiguous faces}
\end{array} \]

3.2 Fast Polygonization Method Based on Marching Cubes Algorithm

Our approach is to create a 256 entry look-up table which is indexed by an eight bit number created from the states of 8 cell corners, and each entry of the table points to a \( 2^n \) entry connectivity table for cell edges which is indexed by a \( n \) bit number created from the states of \( n \) ambiguity faces. There are 656 connectivity tables. The structure of a look-up table is shown in Figure 3.3. Using the disambiguation algorithm described in [11], a \( n \) bit number is created from the state of \( n \) ambiguous faces at runtime. Connectivity tables are constructed as follows.

First, we introduce the indexing convention for the cell vertices, edges, and faces as shown in Figure 3.4. Each of 256 possible configurations is identified by an eight bit number which is constructed by setting bits to 1 if corresponding cell corners have negative function values, otherwise 0. The set of polygons of a given configuration is represented using a connectivity graph of the intersection points on cell edges. For example, if 0th corner and 1st corner of the cell have the negative function value, and the other corners have the positive one, then the configuration is identified by 3 (00000011), and a connectivity graph of a set of polygons (triangles) are 1 \( \rightarrow \) 9 \( \rightarrow \) 3, 3 \( \rightarrow \) 9 \( \rightarrow \) 8. The vertices of a triangle is ordered counter-clockwise. See Figure 3.5.

Since there are two surface vertex connections for an ambiguous face, \( 2^n \) possible configurations of the set of polygons is considered for a cell which contains \( n \)
Figure 3.3: Structure of look-up table

Figure 3.4: Cell indexing convention

Figure 3.5: Configuration 3
ambiguous faces. For example, since the configuration identified by 5 contains one ambiguous corner configuration on 0th face, there are two possible configurations of the set of polygons: $0 \to 8 \to 1, 1 \to 8 \to 10, 2 \to 10 \to 3, 3 \to 10 \to 8$, and $0 \to 8 \to 3, 1 \to 2 \to 10$. See Figure 3.6.

![Figure 3.6: Configuration 5](image)

An ambiguous face is assumed to take the binary state, because there are two surface vertex connections for such a face. We define the state of an ambiguous face: 0 if surface vertex connection is $a \leftrightarrow b, c \leftrightarrow d$, and 1 if $a \leftrightarrow d, b \leftrightarrow c$ in $a \to b \to c \to d$ edge order for inspecting the face as shown in Figure 3.7. To construct a $n$ bit number from the states of $n$ ambiguous faces contained in a cell, each of $n$ possible configurations for such a cell is identified. To retain consistency of surface vertex connection along shared faces, it is necessary to make an edge order for inspecting a face the same between adjacent faces as shown in Figure 3.8. For example the configuration indexed by 26 has 3 ambiguous faces, 0th, 1st, and 2nd, there are $8 (2^3)$ alternatives.
Figure 3.7: Two surface vertex connections for an ambiguous face

Figure 3.8: Inspection order for cell faces
Figure 3.9: Configuration 26
3.2.1 Disambiguation

Assume $\xi_0, \xi_1, \xi_2,$ and $\xi_3$ are function values sampled at corners of the cell face as shown in Figure 3.10. Using the disambiguation algorithm described in [11], we can select the surface vertex connection for an ambiguous face using linear and bilinear interpolation of the function values: $-\xi_1/(\xi_2 - \xi_1)$ and $-(\xi_1 - \xi_0)/(\xi_2 - \xi_3) - (\xi_1 - \xi_0))$. If $-\xi_1/(\xi_2 - \xi_1) < -(\xi_1 - \xi_0)/(\xi_2 - \xi_3) - (\xi_1 - \xi_0))$ then a surface vertex connection represented by solid lines can be selected, otherwise a surface vertex connection represented by dotted lines can be selected.

![Figure 3.10: Local linear and bilinear interpolation on the cell face](image)

3.2.2 Cell Polygonization Procedure

Cell polygonization procedure is listed as follow.

1. Create an eight bit number from the state of cell corners.

2. Create $n$ bit number from the states of ambiguous faces using the linear and bilinear interpolations. If there is no ambiguous face, then 0 is used as $n$ bit number.

3. Find connectivity table using the eight bit number and the $n$ bit number.

4. Create triangles based on the connectivity table.

Since the polygonization method primarily uses a look-up table and only requires function sampling and linear and bilinear interpolations, it only depends on the complexity of a function representing the implicit surface.
3.3 Application

3.3.1 Demos for the Magic Box “Paco”

Using the polygonization method described above, two demos using implicit surfaces were developed for the Magic Box “Pako” [12]. The Magic Box “Paco” is a box-shaped device for interactive art exhibited at Japan Expo in Fukushima 2001. The screenshot of the Magic Box “Paco” is shown in Figure 3.11. Contents for the box are developed using Visual C++ on Windows. Since a three-axis gyro sensor which can detect angles, angular velocities and accelerations is embedded in the box, behavior of the contents can be changed interactively by inclining and shaking the box.

![Screenshot of the Magic Box “Paco”](image)

Figure 3.11: Screenshot of the Magic Box “Paco”

One demonstration shows interactive metamorphosis between a sphere and a torus, and twisting the mixed shape. The following equation is used to achieve the metamorphosis:

\[
sphere \cdot (1 - t) + torus \cdot t
\]

To make the parameter \( t \) bound to one angle, the shape is changed interactively inclining the box. Another angle is used to twist the mixed shape. Figure 3.12 shows the screenshot of the demo. The other demonstration shows convolution surfaces that are changing their shapes according to the angles of the box.
Figure 3.12: Interactive metamorphosis and twisting demo
Chapter 4

Internet-based Interactive Modeler for F-rep Solids

Recently, the Internet has spread around the world, and a large amount of data is being exchanged through the Internet. WWW (World Wide Web) is a good example. The Internet is a powerful tool to share knowledge and to collaborate with many people, in many languages, across many cultures.

The use of interactive visualization of concepts and ideas is one of the keys to the transfer of knowledge. VRML (Virtual Reality Modeling Language) is a well known format to visually describe three-dimensional models and interactive worlds on the Internet. VRML has some disadvantages in that models in VRML are represented using only polygonal meshes. To represent high-quality models, a large number of polygonal meshes are required. It takes a long time to download such models under low-bandwidth Internet connections, and it is difficult to deal with such models on low-end machines. The advantage of using HyperFun is that HyperFun’s mathematical descriptions of complex models are very compact and easy to transmit quickly over the Internet. Furthermore, these mathematical definitions are resolution independent; therefore, they can fit on various size computing systems. In other words, the resolution for visualization of the models defined in HyperFun can be adjusted for machine power.

As discussed in Chapter 2, HyperFun can describe complex models with a small amount of code. HyperFun can be used as lightweight protocol suitable for sharing three-dimensional models through the Internet. Since HyperFun deals with F-rep models, the models can be represented with any resolution.

In this chapter, an Internet-based interactive modeler for F-rep solids is presented.

4.1 Interactive Modeler

4.1.1 Implementation

The modeler is implemented as a Java applet. Java is a cross-platform programming language, and it allows programmers to write applications that can run on web browsers. To visualize three-dimensional models, Java 3D API is used. The polygonization method described in Chapter 3 and HyperFun to Java byte-code compiler used
EmpiricalHyperFun [13] enable interactive modeling. See Figure 4.1 for a screenshot of the Java applet. This is a text-based modeler. The user describes an F-rep model in HyperFun on a text area and polygonizes the model to see the result.

Figure 4.1: Interactive HyperFun modeler applet

To increase the performance and the interactivity of the modeling, a HyperFun program is compiled to Java byte-code. Triangle meshes are generated from a HyperFun program as follows. First, HyperFun to Java byte-code compiler translates a HyperFun program, which is input from the text area into byte-code, and a class is synthesized from the byte-code on the memory. The class contains the method representing the F-rep solid described in the given HyperFun program. Then, using the dynamic class loading mechanism available in Java, the synthesized class is loaded from the memory. Finally the polygonizer generates triangle meshes from the method representing the F-rep solid. A sequence of the translation is depicted in Figure 4.2.
Figure 4.2: Translation sequence from HyperFun program to triangle meshes
Chapter 5

Conclusions

In this thesis, we presented interactive modeling and visualization of F-rep solids with an extendable user interface.

One approach was interactive modeling of convolution surfaces with an extendable user interface. We showed that an extendable user interface, developed by the author, is useful for interactive modeling of functionally-defined objects and that it provides the modeler with flexibility to deal with different types of objects.

We also showed Internet-based interactive modeling of F-rep solids using HyperFun. HyperFun can be used as lightweight protocol for exchanging F-rep models through the Internet. To increase interactivity, a fast polygonization method described in Chapter 3 was developed by the author and is used with the HyperFun to Java bytecode compiler in the Java applet.

We plan to create an Internet-based interactive modeler for general F-rep solids with an extendable user interface combining the extension mechanism of the modeler for convolution surfaces presented in Chapter 2 and the Java applet for modeling F-rep solids presented in Chapter 4.

To extend the Java applet, we also plan to construct a collaborative functionally-based shape modeling environment on Sparkle Gate [14]. Sparkle Gate is an open and free integrated communication site. Thousands of people have been registered at this site. There are various services based on a chat system such as games, diary, bulletin board system, and so on. The chat system is useful for Internet-based education of functionally-based shape modeling.
References


