Interactive Modelling of Convolution Surfaces with an Extendable User Interface

Yuichiro Goto[*], Alexander Pasko[&]

([*]) Graduate School, University of Aizu, Aizu–Wakamatsu, 965–85–80 Japan.
m5041110@u-aizu.ac.jp
([&])Department of Digital Media, Hosei University, Koganei, 184–8584 Japan.
pasko@k.hosei.ac.jp

Abstract

Convolution surfaces enable the user to model complex free-form shapes. Due to analytical solutions for some kernel functions and skeletal elements, it is possible to model convolution surfaces interactively. An extendable user interface allows the user to design models using different types of convolution surfaces. New primitives can be easily bound to the modeller using the proposed binding technique. Models generated in HyperFun language can be exchanged between modelling tools on several platforms.

1. Introduction

Convolution surfaces are a class of implicit surfaces 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 modelling.

In this paper, interactive modelling of convolution surfaces with an extendable user interface is presented. Using an extendable user interface, the modeller is not bound by limited types of convolution surfaces. The user can introduce new primitives in the modeller without a system source code level extension. Since there are different types of kernel functions and skeletal elements, many various types of convolution surfaces exist. For this reason, an extensibility of the modeller is very useful.

The modeller exports created models in the form of HyperFun programs [4, 6]. HyperFun is a high-level language used for describing F-rep models [5]. F-rep is a model, which defines geometry of objects in the form of $F(x1, x2, ..., xn) \geq 0$. F-rep is more general than traditional implicit surfaces. There are several advantages of using HyperFun as the output. HyperFun serves as an exchange protocol: the models in HyperFun can be transfered between different tools. Various modelling systems supporting HyperFun have been developed on several computer platforms such as Unix, Linux and Windows. Since HyperFun is designed to be very simple, the user can easily master it.

2. Convolution Surfaces

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

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

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

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

A skeletal element defines a shape 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.

3. HyperFun Language

HyperFun is a high-level language designed for specification of functionally based models (F-rep) in the form of $F(x1, x2, ..., xn) \geq 0$. HyperFun is simple, but has enough facilities for creating complex F-rep models [4]. A function defining an F-rep object can be constructed using assignment, conditional selection, and iteration statements. Conventional arithmetic and relational operators, standard mathematical functions, built-in set-theoretic operators, and system F-rep library functions are available. The user can extend the library on different language levels (HyperFun, C, Java). The convolution surface primitives are examples of F-rep library extensions on C level. The subject of this work is the interactive modeller extension without its reprogramming.

4. Modeller with an Extendable User Interface

MAM/VRS [7] is used for implementing the modeller. 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, Linux, and Windows.

The screenshot of the modeller is shown in Figure 1. The modeller 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 modeller using four windows, but an operation such as moving vertex are not allowed in the perspective window.

The modeller is extendable by convolution surface primitives, which become built-in primitives of the modeller. If convolution surfaces using new kernels are added to the F-rep library, it is possible to use those primitives without rewriting the source code of the modeller. The modeller is different from usual modellers in this aspect. Primitive bindings are specified in an initialization file. The modeller should be invoked with this file. Available built-in primitives of the modeller 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.

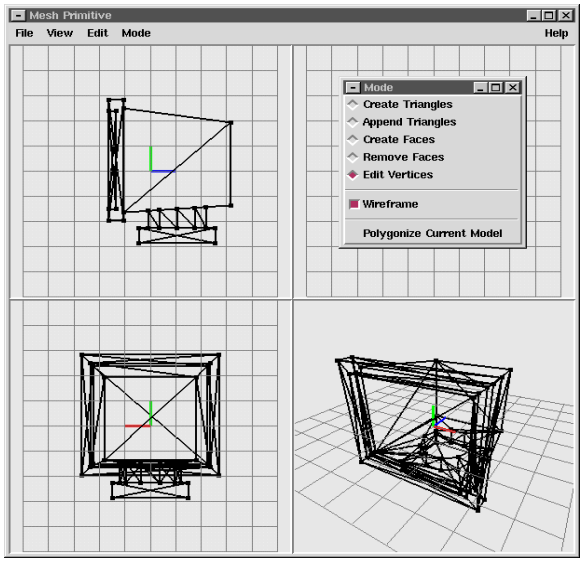

Figure 1: *Screenshot of the modeller*

Figure 2: *Polygonized surface*

Now, we give an outline of modelling procedures using the modeller. First, a file for binding convolution surface primitives should be prepared. Next, the modeller should be invoked with the file. If the file has no syntax error, the modeller's interface appears on the screen. If default values are used, some objects may be put on the working areas of the modeller. Then, the user interacts with the modeller to create and manipulate the objects. During the interaction, the user can obtain a polygonized surface for the current model. Figure 2 shows a polygonized surface. Finally, the modeller generates a HyperFun program for the created model.

4.1. Binding primitives

This section describes an initialization format for primitives binding. This file is important to inform the modeller what F-rep primitives of convolution surfaces are used, the way to display them during interactive modelling and how to generate HyperFun programs for them. Primitives

are specified in a function form, that is like $hf < name > (par_1, par_2, ..., par_k)$. The function form is the same as one used in HyperFun. The binding is achieved by passing parameters of convolution surface primitives to built-in primitives of the modeller. If bindings are successful, the modeller 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 3 is an example of the initialization file.

```
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 3: *Initialization file example*

4.1.1. 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. The type of the F-rep element is "primitive" or "operation", but now only "primitive" can be specified. Operations are such as blending union, scaling, and so on. An F-rep primitive is declared with its function used in HyperFun. Parameter "x" is reserved in HyperFun

```
HFModel {
  -- textual information --
  "my primitive"
  -- type --
  primitive
  -- F-rep primitive --
  hf<name>(x, center, b, c, d);
}
```

4.1.2. 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 modeller. 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}
}
```

4.1.3. 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. --
}
```

4.1.4. Visual

In this block, a built-in primitive of the modeller 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”, which means the use of built-in primitives. We plan to use other types such as “vrml” and “dxf” for externally produced polygon models. The modeller will import those files and use them like built-in primitives. 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 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 4: *Initialization file for the body of the fish*

5. Example

In this section, an example is given using the modeller. Figure 9 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.

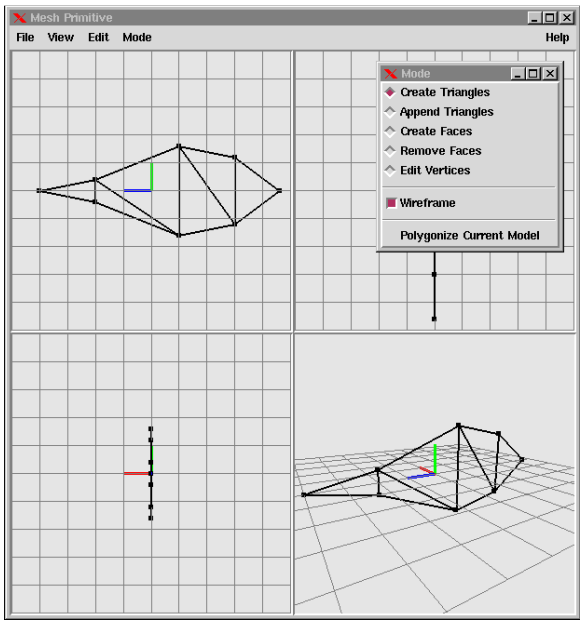

Figure 5: *The modeller with some triangles*

Figure 4 is an initialization file for the body of the fish. Only triangles are used for the body. Figure 5 is a screenshot of the modeller invoked with the file in Figure 4. Triangles are added on the working area of the modeller. To see a precise shape of a current model, polygonization is useful. Figure 6 is the result of polygonization of the model in Figure 5. Figure 7 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 8. The model contains 13 triangles and 10 lines. Polygonization is done on a SGI O2 workstation with 180 MHz MIPS R5000 processor. Table 1 shows polygonization time for single line and triangle, and fish model in Figure 8.

single line	4.97 sec
single triangle	6.43 sec
fish model	77.01 sec

Table 1: *Polygonization time*

6. Conclusions

In this paper, interactive modelling of convolution surfaces with an extendable user interface was presented. Now, extensibility of the modeller is limited by convolution surface primitives. However, based on this modeller’s features, more powerful modellers can be developed to extend the list of primitives and operations with arbitrary types. The complete

fish model in HyperFun is only a 2.3K file. Using HyperFun as a lightweight protocol between the interactive modeller and the visualization software (polygonization or ray–tracing) will allow us to implement a Java version of the modeller running under a Web browser at the client side.

Acknowledgments

The authors would like to thank Ken Yoshikawa, Eric Fausett, Carl and Jody Vilbrandt for their help and support.

Appendix

```
-- The HyperFun code of the bone --

head(x[3], a[1])
{
  array vertex[9];
  array index[3];

  vertex = [0, 7, -14, 0, 0, -21, 0, -7, -
14];
  index = [1, 2, 3];
  r = 1.5;

  head = hfConvMeshR(x, vertex, index, r);
}

body(x[3], a[1])
{
  array begin[27];
  array end[27];

  begin = [0, 0, -15, 0, -7, -11, 0, -
9, -5, 0, -7, 1, 0, -3, 7, 0, 8, -
2, 0, 5, 4, 0, 8, -8, 0, 2, 10];
  end = [0, 0, 15, 0, 7, -11, 0, 9, -
5, 0, 7, 1, 0, 3, 7, 0, -8, -2, 0, -
6, 4, 0, -8, -8, 0, -2, 10];
  r = 0.6;

  body = hfConvLineR(x, begin, end, r);
}

eye(x[3], a[1])
{
  array begin[3];
  array end[3];

  begin = [5, 0, -17];
  end = [-5, 0, -17];
  r = 0.75;

  eye = hfConvLineR(x, begin, end, r);
}

my_model(x[3], a[1])
{
  my_model = head(x, a) | body(x, a) \ eye(x, a);
}
```


References

1. J. Bloomenthal and K. Shoemake. Convolution Surfaces. *Computer Graphics (Proceedings of SIGGRAPH '91)*, **25**(4):251–256, 1991 1
2. A. Sherstyuk. Convolution Surfaces in Computer Graphics. *Ph.D. thesis*, Monash University, Australia, 1999, 123 pp. 1, 4
3. J. McCormack and A. Sherstyuk. Creating and rendering convolution surfaces. *Computer Graphics Forum*, **17**(2):113–120, 1998. 1, 4
4. V. Adzhiev, R Cartwright, E. Fausett, A. Ossipov, A. Pasko and V. Savchenko. HyperFun project: a framework for collaborative multidimensional F–rep modelling. *Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop*, J. Hughes and C. Schlick (Eds.), pages 59–69, 1999. 1, 2
5. A. A. Pasko, V. Adzhiev, A. Sourin and V. Savchenko. Function representation in geometric modelling: Concepts, implementation and application. *The Visual Computer*, **11**(8):429-446, 1995. 1
6. HyperFun, http://www.hyperfun.org 1
7. MAM/VRS, http://wwwmath.uni–muenster.de/informatik/u/mam/ 2
8. Tcl/Tk, http://www.scriptics.com 2

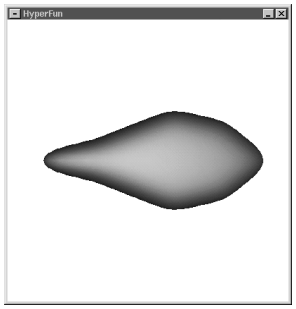

Figure 6: *Polygonized model of the body of the fish*

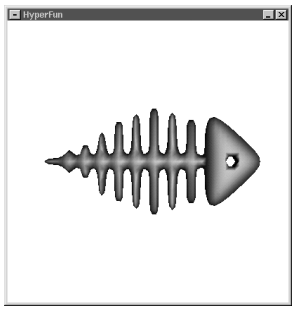

Figure 7: *Polygonized model of the bones of the fish*

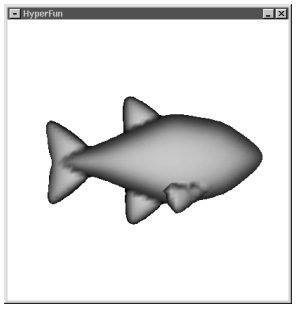

Figure 8: *Fish image*

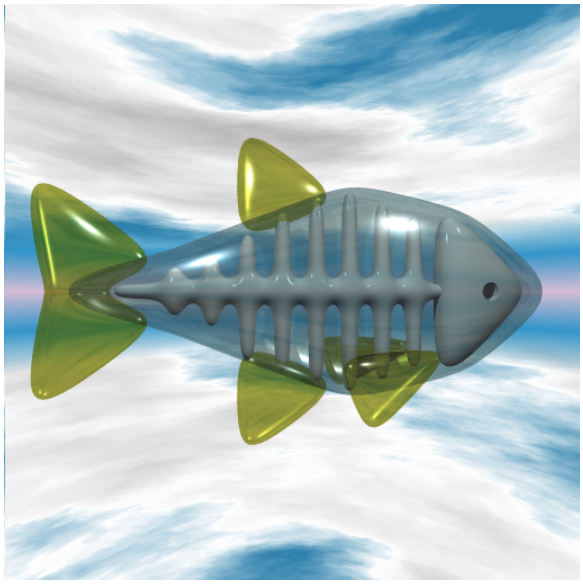

Figure 9: *Ray-traced fish image*