Preliminary programming for librarization of Haptic Primitives based on constructive solid geometry and god-object

Do-Hyung Jin, Ki-Uk Kyung, Dong-Soo Kwon

Dept. Mechanical Engineering, KAIST, Daejeon, Korea
(Tel: +82-42-869-3042; Fax: +82-42-869-3210; E-mail: {jindh, kyungku, kwon}@robot.kaist.ac.kr)

Abstract: We propose *'the haptic primitive'* for haptic rendering without the need to solve complicated parametric equations. To develop 'the haptic primitive', we adopted "the God-Object Method" as a haptic rendering algorithm and applied 'Constructive Solid Geometry' to manage haptic objects. Besides being used in the 'ghost library' of PHANToM^{TM} our method can be used as a basic component for developing tools and libraries that aim to simplify haptic modeling. It can also be applied to tactile display modules and temporal display modules. Ultimately it can be developed into a one-stop haptic modeling tool that enables the user to more conveniently create a tangible CAD systems or a tangible e-commerce system.

Keywords: haptic, haptic primitive, CSG model, god-object, haptic modeling, haptic CAD, haptic library, tangible environment, virtual environment

1. INTRODUCTION

We hope to combine various haptic elements to enable the user to interact with visual feedback and force feedback in virtual environment. Firstly, we need an appropriate haptic interface device to generate enough force and torque to simulate physical properties. Secondly, for haptic rendering, we need system architecture that consists of a low-level control algorithm, collision detection, force-response algorithm and so on is also necessary. Thirdly, we need to investigate haptic rendering that computes the force relationship between a virtual object and a haptic device. More over we need objects that have physical properties rendered by a haptic rendering algorithm and we also need a graphic rendering algorithm to graphically render those objects [1].

Some kinds of tools that easily help the user to establish a tangible virtual environment have not been suggested yet, because a perfect and easy method of integrating a haptic model and a graphic model has not been completed. Therefore, to render objects haptically and graphically in the same time, programmers must define the force relation and solve complicated parametric equations at every part of a virtual environment; they must program all of the result.

We present 'the haptic primitive' that can be extended to a development tool. The haptic primitive helps the user to more easily design haptic environment with the aid of the graphic user interface. The user can avoid the complicated process of combining visual and haptic rendering. Take for example, the user who wants to confirm the state of connection or the shape of an environment by touching it. While developing a virtual environment with software for the 3-D environment development, the user can use 'the haptic primitive' to immediately sense forces through a haptic mouse. We hope to make our algorithms as libraries that can be applied to any haptic devices.

To develop 'the haptic primitive', we adopted 'the God-Object Method' as a haptic rendering algorithm [2]. We need this method because the usual 'Vector Field Method' induces the confusion that when transitions are made accidentally. Furthermore, under the vector field method, force discontinuities can be encountered while traversing volume boundaries and small or thin objects need to generate convincing constraints. We therefore applied 'constructive solid geometry (CSG)' to manage objects in the developed environment. The CSG model is a good metaphor for constructing complex objects. Moreover, it can be created by applying Boolean operations on simple, mathematical objects [1,3]. The famous company, SensAble, produces its own haptic library as well as PHANToM^{TM} haptic device [4]. SensAble's GHOST libraries are provided for the developers who use PHANToM^{TM} to create tangible environments. However, the company does not provide a convenient and intuitive user interface.

To enable the user to easily adjust to the environment, we modeled our graphic user interface after the basic frame of the graphic user interface of commercial software used in the development of 3-D environments. As a result, we packaged 'the haptic primitives' that constituted the haptic environment in the form of a library that could easily be connected to other programs and devices. The package can be used as a basic component for developing tools and libraries that aim to simplify haptic modeling. It also can be applied to tactile display modules and temporal display modules, and ultimately developed into a one-stop haptic modeling tool that enables the users to more conveniently create a tangible CAD system or tangible e-commerce system.

In Section2, we define haptic rendering and CSG trees, as well as our approach to developing haptic libraries. In Section3, we explain the algorithms in detail. In Section4, we discuss the applied results of the algorithm, and in Section5 we summarize our conclusions.

2. HAPTIC PRIMITIVE

To this point there have been two different haptic approaches.

The first one is a bottom-up approach.

Defined as polygon-based, and this approach starts from small components, and moves on to the complete the whole object. In other words, when modeling an object, we start by using a 3D modeling tool to show the object with polygons. We then use individual program codes to apply the haptic.

The second approach is a top-down approach. Defined as parameter-based, this approach divides an object into smaller pieces and applies a haptic to the object. Unlike the first approach, the top-down approach starts from an assumption that it would be easier to show the objects with a parameter

equation.

With the top-down approach, the haptic primitive will be defined. Because the haptic primitive is directed at the top-down approach, the user might take the risk of solving complex equations while programming to break down the whole object into smaller pieces. To prevent this break-down, a primitive based on a CSG model is provided. With the CSG-based model, we can express where the polygon is located with an x-y-z parametric equation.

For example, to show where a box is located in relation to a haptic the following equations have to be stored.

$$Box = \{A \le x \le B, C \le y \le D, E \le z \le F\}$$
$$Sphere = \{x^2 + y^2 + z^2 \le A\}$$

When a primitive is provided in this way, then with only a few constants, the user can express the form of the primitive, as well as the location of the constraints. If the primitive is a box, six constants can define the constraints, and three constants can show the center or the starting point. If the primitive is a sphere, one constant can define the volume (so, it is a radius!) and three constants can define the center.

	Position	Edge or Surface	Total
Box	3	2*3	9
Sphere	3	1	3
Primitive	3	2*n or 1	O(n)

Table 1. The number of constraints

We can therefore see that with the top-down approach, the task is vastly reduced, compared to the bottom-up approach.

The expression is very limited when the object is simply broken down to components.

With the CSG model, however, we can conduct minus operations or intersection operations. Therefore, if the user saves up the previous results, and loads them, there will not be a problem to overcome the limitation.

In PHANToM there is already a library called GHOST, which is taking over this role. Because GHOST is a commercial software, it has many more functions. It has the form of DLL-based implementation, and cannot be applied to other devices. Our objective, therefore, is to suggest a library with a haptic primitive that can be applied to all haptic devices.

3. TECHNICAL DETAIL

A haptic primitive is attempted when Raymaeker has the experience of realizing haptic rendering based on haptic primitive with CSG model. [5]
We implemented a haptic primitive based on the attempt.

3.1 Primitive

Our primitives need to have the form of class that can be easily added to and adjusted. We therefore tried to save the encapsulation by defining the class of primitive and adding methods for approaching core data.

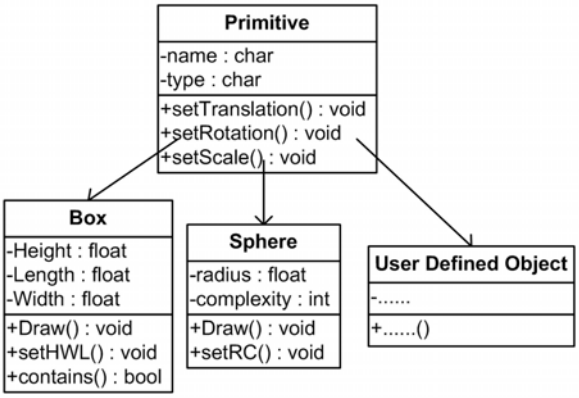

Diagram 1. Class structure

We designed the structure of haptic primitive as diagram 1.

According to the structure in diagram 1, we can write pseudo code of the primitive class as following algorithm 1.

```
Class Primitive {
    void setTranslation(GLfloat xt, GLfloat yt, GLfloat zt);
    void setRotation(GLfloat xr, GLfloat yr, GLfloat zr);
    void setScale(GLfloat xs, GLfloat ys, GLfloat zs);
    string name;
    // 'b': Box, 's': Sphere, 'c': cone, 'y': cylinder
    char type;
}
```

Algorithm 1. Primitive class definition

According to the structure in diagram 1, we organized two (box, sphere) of the sub-classes as following algorithms 2,3.

```
Class Box: public Primitive {
        void Draw();
        void setHWL(int h, int w, int l);
        bool contains(float x, float y, float z);
        float height;           // y-axis
        float width;            // x-axis
        float length;           // z-axis
        float point_head1[3];
        float point_head2[3];
        float point_head3[3];
        float point_head4[3];
        float point_foot1[3];
        float point_foot2[3];
        float point_foot3[3];
        float point_foot4[3];
}
```

Algorithm 2. Box class definition

```
Class Sphere: public Primitive
{
        void Draw();
        void setRC(int r, int c);
        bool contains(float x, float y, float z);
        float radius;
```

```
        int complexity;
}
```

Algorithm 3. Sphere class definition

These classes of above algorithms have two features. First, it is easy to add primitives. Because Draw() or Touch() has been inherited with a virtual function, any form of primitive can easily be added. The features of each object can therefore be realized easily. Second, each feature can also be added or deleted. Regardless of the device, it is easy to add physical features can, and delete unnecessary functions.

3.1.1 Scene Tree

Our program uses the concept of a scene tree, which is a technique of the open inventor™. Although the scene tree enables us to effectively manage many objects in the virtual space, we combined it with CSG model due to our haptic need.

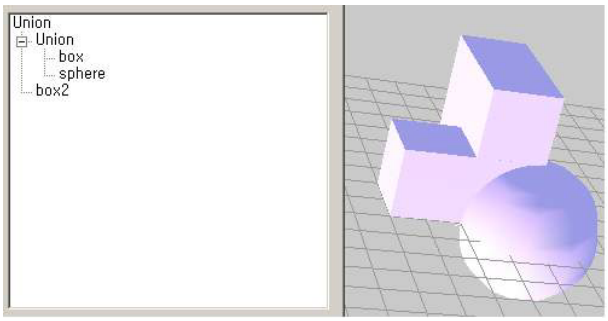

Figure 1. The scene tree

Figure1 shows our own developing scene tree.

$$Tree = Leaf \mid [Tree(Left): \quad Internal \quad :Tree(Right)]$$

The scene tree that we used is a binary tree with internal nodes and leaf nodes. The leaf node has the attribute "Primitive." And internal node has the attribute "Operation."

3.2 Operation

Although we tried to support all three operations of the CSG model, we could only support the union operation. However, because the other concepts of the CSG model are similar to the union operation, we soon hope to support the other operations (namely, the intersection and subtraction operations).

3.2.1 Intersection

Raymaeker explained how to implement the intersection operation. [5]

We now explain how to add physical characteristics to the intersection operation.

```
GetIntersectionSurface() {
    // Find where the Surface Contact Point exists
    SCP = GetIntersectionSCP();
    // Set the surface property
    SetSurfaceProperty(SCP→Surface, intersection);

    return machine_state;
}
```

Algorithm 4. Intersection algorithm

The intersection implemented in algorithm 4 suggests a way to add the physical texture, which was missing in Raymaeker's works.

The texture added to the intersection refers only to the physical surface, though the intersection is an operation that also considers the inside volume.

A structure is therefore needed to determine if the added physical characteristic is related to the volume or the surface.

We can determine if the texture property that is related to the surface by results of operations about surfaces. If the SCP detects the surface while calculating the intersection, we can simultaneously calculate the intersection operation and express the physical property of texture.

In this way, we can present other diverse physical features.

3.2.2 Union

We now explain how to add physical characteristics to the union operation.

```
GetUnionSurface() {
    // Find where the Surface Contact Point exists
    SCP = GetUnionSCP();
    // Set the surface property
    SetSurfaceProperty(SCP→Surface, union );

    return machine_state;
}
```

Algorithm 5. Union algorithm

The implementation of the union operation is similar to that of the intersection operation. Through the implementation of the union operation, we can add a physical property such as texture and we can also support other physical properties.

As with the intersection operation, the point of the union operation reveals the property of the surface on which the SCP occurs in real-time

3.2.3 Subtraction

Before the subtraction, we made some assumptions. Similarly to the paper of Raymaeker, we assumed that all of the objects were solid. The subtraction operation can make a new surface appear. To give a new surface a physical property, we assumed that the entire new surface had the same physical property as its parent surface.

Of course, we enable the user to assign a physical property to a new surface. The optional properties of the objects are the subtrahend and the minuend.

The rest of our subtraction implementation is similar to

Raymaeker's subtraction implementation [5].

However, we added a method that could support the expression of physical properties.

```
GetSubtractionSurface() {
    // Find where the Surface Contact Point exists
    SCP = GetSubtractionSCP();
    // Inherit the surface property to a new surface
    if (SCP→Surface is new?)
        SetProperty(SCP→Object, SCP→Surface)
    // Set the surface property
    SetSurfaceProperty(SCP→Surface, subtraction);

    return machine_state;
  }
```

Algorithm 6. Subtraction Algorithm

The subtraction implementation is the same as the union implementation.

Of course this implementation also supports other physical properties.

4. RESULTS

For the CSG algorithms, we used the e-Touch [6] library. For the haptic device, we used a device called PHANToM Premium 1.5A. This device was attached to a Pentium IV, 1 GHz computer with 512 MB RAM, running Windows 2000. We used OpenGL to render the graphical output.

Figure 2. Haptic feedback system with PHANToM™

Figure 2 shows our implemented results. The user can construct the virtual environment using sphere and cube primitives. The user can control the size, stiffness and position of spherical and cubic primitives. And then the user applies the intersection, union or subtraction at will.

The test on PHANToM™ was successful and it enables the user to simultaneously design and touch the object.

In the near future, we will experiment on our haptic device – the haptic mouse, which is shown in Figure 3 and make a library for haptic mouse. We are trying to apply proposed algorithm to this device. Because the interface contact and grabbing force feedback of our haptic device mouse is similar to that of PC mouse, we expect good results from the experiment.

One advantage of the proposed algorithm is that it was not designed for specific application hardware. When the user matches the IO channels, the algorithm can work well.

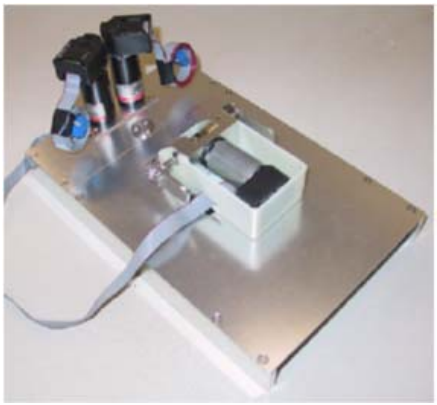

Figure 3. Haptic mouse system

5. CONCLUSIONS

This paper shows preliminary programming for the liberalization of haptic primitives based on constructive solid geometry and god-object. We used haptic primitive because they are necessary for intuitive haptic rendering. To realize our goal, we used algorithms based on constructive solid geometry and the god-object Method.

This algorithm suggests that a haptic modeling tool can be made as an extension of haptic primitives. It enables users to be able to establish the haptic virtual environment by just clicking an object without the burden of complicated programming. Finally, we have experimentally confirmed that the haptic primitive algorithms can drastically decrease the programming load for making a virtual environment.

A GHOST library already exists for PHANToM. Compared to GHOST, our proposed primitives provide an intuitive user interfaces and data structure that contains tactile information as well as force feedback.

In the near future, we plan to develop haptic primitives with a tactile display or with our novel haptic mouse.

Haptic primitives can be applied to 3D CAD (Computer Aided Design) software. The part assembly is a possible application because of haptic sensation can be used in multimodal simulation area. The user can assemble the parts while feeling the contact force. The haptic mouse helps the user to determine whether the parts can be assembled or not. In addition, the touch sense can help the user determine a suitable clearance of assembled parts. Our haptic primitive helps the developer to realize this application.

ACKNOWLEDGMENTS

This research was supported by the 'Smart Haptic Interface Project (2004-S-030, Korea Ministry of Information and Communication)' and KOSEF(Korea Science and Engineering Foundation).

REFERENCES

[1] K. Salisbury, F. Conti, F. Barbagli., "Haptic Rendering:

Introductory Concepts", IEEE Computer Graphics and Applications, May–April 2004.
[2] C. Zilles and K. Salisbury: "A Constraint Based God-Object Method for Haptic Display", Master Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1995.
[3] N. Steward, G. Leach, and S. John. An improved Z-buffer CSG rendering algorithm. In 1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware, pages 25–30, Lisbon, PT, August 31–September 1 1998.
[4] SensAble Technologies, http://www.sensable.com
[5] Chris Raymaekers, Frank Van Reeth. Algorithms for Haptic Rendering of CSG Trees. EUROhaptics 2002.
[6] e-Touch Project, http://www.etouch3d.org/
[7] F. Conti, F. Barbagli, D. Morris., "The CHAI Libraries." EUROhaptics 2003.
[8] T. Massie. A tangible goal for 3d modeling. IEEE Computer Graphics and Applications, 18(3):62–65, May–June 1998.
[9] J. K. Salisbury. Making graphics physically tangible. Communications of the ACM, 42(8):74–81, August 1999.
[10] N. Steward, G. Leach, and S. John. An improved Z-buffer CSG rendering algorithm. In 1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware, pages 25–30, Lisbon, PT, August 31–September 1 1998.
[11] Immersion Corporation, http://www.immersion.com
[12] SGI, http://www.sgi.com