Development of Virtual Objects of Arbitrary Shape

by

Hans David Høeg

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

© Massachusetts Institute of Technology 1994. All rights reserved.

Acceptance

J. Kenneth Salisbury, Jr.
Principal Research Scientist
Thesis Supervisor

Peter Griffith
Chairman, Departmental Committee on Graduate Students
Development of Virtual Objects of Arbitrary Shape

by

Hans David Høeg

Submitted to the Department of Mechanical Engineering
on May 6, 1994, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Mechanical Engineering

Abstract

In this thesis I describe the development of models for the simulation of arbitrary geometric shapes to be used in a 2-D virtual force reflecting environment. After developing a set of models necessary for the construction of functional shapes, virtual building blocks, called haptic primitives, were created. The blocks were then joined to form composite objects of varying geometry. The haptic primitives work fairly well, allowing an environment builder to construct an array of different structures. The potential application of haptic primitives is very large, and with proper program optimization it is possible that highly complex environments can be created. This could prove very valuable to a field such as medical micro-surgery, or to a field such as engineering design, where it is conceivable that designs could be materialized by transforming a mesh of coordinates into a mesh of force reflecting surfaces, thereby providing the designers with the feedback of touch.

Title: Principal Research Scientist
Acknowledgments

I would like to thank the following people for helping me complete this thesis:

Dr. J. Kenneth Salisbury, for giving me the opportunity to undertake such an incredibly fun project. Despite being buried in work himself, his door was literally always open. His advice and support are greatly appreciated.

Thomas H Massie, without whom virtual reality and many other things would be a giant step behind where they are today, including this thesis! Although being endlessly busy, he was always there to help, whether it was supplying computers, creativity, incentive or just plain support. Thanks Thomas, for being one of the best friends anyone could ever have.

Hans Jacob S. Feder, who without hesitation brainstormed, programmed, drew, wrote and won the battle with \LaTeX{} for me. His ubiquitous support and advice are greatly appreciated, and his friendship is one of a kind.

Most importantly, my father, and mentor, who’s 23 years of devotion and guidance has kept me pretty much out of trouble!

The author would also like to gratefully acknowledge the support of the Naval Training Systems Center contract numbers 60326 (VETT Haptics) and 60678 (Tool iIhandle Haptic Interface).
Contents

1 Introduction ........................................... 7
  1.1 Overview ........................................... 8

2 The Nature of Virtual Objects ....................... 9
  2.1 Force Reflection ..................................... 9
    2.1.1 Debugging ...................................... 10
  2.2 Erroneous Ejection ................................. 10
  2.3 Corners ........................................... 11
    2.3.1 Methods for Modeling Virtual Corners ........ 12

3 Planes and Functions ................................ 15
  3.1 2-D Planes ......................................... 15
    3.1.1 Plane Segments ................................. 17
  3.2 Mathematical Functions ............................ 18
    3.2.1 Analytical Solutions .......................... 18
    3.2.2 Numerical Solutions: Function Segments ...... 20

4 Building Blocks ....................................... 22
  4.1 Four Plane Blocks .................................. 23
  4.2 Blocks with Function Surfaces .................... 24
  4.3 The Odd-Parity Rule ................................ 25

5 Virtual Composite Objects ............................ 27
  5.1 Joining Blocks ..................................... 27
List of Figures

2-1  The problem of computing and directing the reaction force. . . . . 10
2-2  Dividing a solid into force reflecting regions, I - IV. . . . . . . 11
2-3  Force reflection from corners. . . . . . . . . . . . . . . . . 12
2-4  The problems with very sharp geometries. . . . . . . . . . . . 13
2-5  Touching a sharp concavity. . . . . . . . . . . . . . . . . 14
2-6  Corner created with a corner block. . . . . . . . . . . . . . . 14

3-1  Plane and its normal vector. . . . . . . . . . . . . . . . . 16
3-2  Semi-infinite block. . . . . . . . . . . . . . . . . . . . . . 18
3-3  Moving around inside an object. . . . . . . . . . . . . . . 19
3-4  Vector from the PHANTOM point to a mathematical function. . . . . 19
3-5  Numerical solution of shortest distance to a function segment. . . 20

4-1  Building block defined by 4 planes. . . . . . . . . . . . . . 23
4-2  Block with mathematical function as force reflecting surface. . . 24
4-3  Odd-Parity Rule for determining the interior of a polygon. . . . 25

5-1  Testing for proximity of PHANTOM point. . . . . . . . . . . 27
5-2  Composite object. . . . . . . . . . . . . . . . . . . . . . 28
Chapter 1

Introduction

In his undergraduate thesis Thomas H. Massie[3] created a system which provides the necessary force feedback to allow users to interact convincingly with artificial or remote environments. The system, a three degree of freedom force reflecting haptic interface called The PHANToM, makes it possible for users to "touch" and "feel" virtual objects. The device works extremely well, as users are able to discriminate between rough and smooth surfaces and between objects of varying geometry and compliance. The potential applications for The PHANToM are virtually unlimited, and they hinge upon the construction of realistic virtual environments. Much of the future of this science, therefore lies in developing software which effectively can model the real world.

In this thesis I addressed the creation of randomly shaped polyhedral objects in 2-D. Specifically, the purpose of the endeavor was to develop algorithms that can build virtual objects of arbitrary shape, thus taking a step forward from the basic virtual objects, called haptic primitives, such as blocks oriented parallel to the axes of the workspace. A key factor was the creation of virtual building blocks, much like Legos or bricks, which when assembled appropriately can construct composite objects with a wide range of geometries.
1.1 Overview

Chapter 2 gives a short background of the nature and sophistication of the virtual objects which existed when I began my research. It addresses some of the fundamental and recurring problems encountered in the modeling of real world shapes and solids.

Chapter 3 treats the development of walls which are oriented arbitrarily with respect to the axes of the workspace. Chapter 4 goes on to examine the creation of building blocks with various thicknesses and stacking abilities. In chapter 5, I address the construction of arbitrary geometric shapes with the block primitives developed in chapter 4. Finally, possible improvements and applications of the haptic primitives are discussed.
Chapter 2

The Nature of Virtual Objects

2.1 Force Reflection

A virtual object, called a haptic primitive, is a computer generated geometric shape or solid intended to simulate a physical object. When its surface is penetrated by the PHANTOM point, which is a mapping of the spatial position of a PHANTOM haptic interface onto virtual space, a physical force is exerted on the interface. In order to simulate real surfaces, one must have an appropriate model for the reaction force that is generated. From the basic principles of mechanics it is found that the reaction force experienced when interacting with solid object is a function of its geometry and its molecular structure. The molecular structure determines the stiffness and frictional properties of the solid, and it is these properties we are interested in when modeling objects. To begin with we will look at the concept of stiffness, which determines the compliance of a solid.

Experimentation with an array of different physical objects taught us that their compliances in many cases can effectively be modeled as mechanical springs. The reaction force exerted by the depressed surface of an object is then given by the following equation:

$$F = -kx$$

(2.1)
where \( x \) is the amount of depression, and \( k \), the spring constant, represents the stiffness of the spring. Moving to virtual objects, we found it most effective to model them as non-compressive, i.e. treating their surfaces as fixed with respect to each other. This means that the PHANToM point is allowed to enter the volume of the solid, and that the force thus generated is proportional to the depth of penetration, \( x \), and the stiffness, \( k \), of the solid.

2.1.1 Debugging

Software procedures very seldomly work as intended the first time (!). This being the case, it is often nerve racking to test a new algorithm with the PHANToM in force reflecting mode. If the PHANToM point is within an object when the simulation begins, a force of up to 10 Newton's suddenly can be exerted and possibly damage the PHANToM. An algorithm that draws a vector with origin in the PHANToM point, symbolizing the direction and the magnitude of the force that would be exerted if the PHANToM were in force reflecting mode, was therefore developed. This allows the programmer to test new code while the PHANToM force is turned off.

2.2 Erroneous Ejection

![Figure 2-1: The problem of computing and directing the reaction force.](image)

A virtual object can be entered from many directions. The orientation of the reaction force vector therefore has to change according to the direction from which the PHANToM point entered the object. Although an object can have many different
sides, it only has one volume. Thus, when the PHANToM point has entered the object, it is always in the same volume. It could however have entered this volume from any direction. Consequently a problem arises in deciding which way to direct the reaction force, as illustrated in figure 2-1.

Theoretically, the most correct method for dealing with this problem is to keep a time history of the PHANToM point’s trajectory. Research on this is currently being done[4].

![Diagram showing force reflecting regions I, II, III, and IV.](image)

Figure 2-2: Dividing a solid into force reflecting regions, I - IV.

The other method for dealing with this problem, called the Vector Field Method, consists of dividing the volume into regions. Each region will then have a part of the object’s surface belonging to it. When entered by the PHANToM point, a region will produce a reaction force normal to its corresponding surface.

This region method works fairly well, but fails when the PHANToM point penetrates the object in such a way that it departs the original region it entered. The direction of the reaction force then changes, in which case the PHANToM point can then be shot out through a different surface from the one which it originally crossed. This is especially a problem with corners, where the PHANToM point need not penetrate the object very deeply before it finds itself in a new region.

### 2.3 Corners

In programming force reflecting surfaces the modeling of corners is a recurring problem. Corners present problems because they are discontinuities in the surface to-
pography of an object. Experimentation with touching physical corners only gave limited feedback on what kind of force model should be used in creating virtual corners. When touching a corner with a finger it is difficult to determine the direction of the reaction force. Consequently, it is not obvious what instructions should be given to the computer when the PHANToM point is interacting with a corner. There are several ways to treat corners, and the model used will depend on the type of surface you are trying to create and the feel you are looking for. In the following we will look at some of the corner models that were implemented, along with their respective benefits and drawbacks.

2.3.1 Methods for Modeling Virtual Corners

![Figure 2-3: Force reflection from corners.](image)

In using the region method discussed above, the problem arises when the PHANToM point is right on the borderline between two regions, which is a common situation when interacting with corners. This is shown in figure 2-3. The computer consequently has a "hard time" deciding which way to direct the force, and we get an unstable condition where the force changes directions very rapidly, oscillating between two extremes. This kind of discrete behavior is obviously not characteristic of real world objects, which when acted upon by a force such as human touch, will produce a single resultant force which direction varies continuously with the change
of position of the external force source. Despite this fact, the model was found to work very well with convex corners.

Another way of modeling a corner is to create an overlapping region, in which the surfaces used to create an object overlap each other. One thus gets a region in which several regions of the object can be active, i.e. contain the PHANTOM point simultaneously. The resultant force is then a vector sum of the forces generated from the penetration into each region. This proved to give a very smooth feel, where the force vector changed direction continuously as one moved across the corner.

![Diagram](image)

Figure 2-4: The problems with very sharp geometries.

The problem with the overlapping region method arises when one tries to model sharply concave corners. It was found that sharp geometries are very soft. The reason for this is the small resultant force that is produced when adding the force vectors for each region. One can see that although the individual force vectors generated by each region can be quite large, their directions are strongly divergent, and the resultant ends up being small, as shown in figure 2-4. As the sharpness of the corner gets very large, the resultant vector actually becomes negligibly small and vanishes in the limiting case. Again, this kind of behavior is not true of a real object. There are some ways to deal with this problem. Since we are simulating human touch, it would be logical to operate with the same constraints that are presented by the real world. For example, it is not possible for the human finger to touch sharply concave cavities. The finger will be obstructed by the walls of the notch as it attempts to penetrate beyond a certain depth, as shown in figure 2-5.

A third way of modeling corners is to construct special corner blocks. One can for
example use blocks similar to circle sectors and wedge these between other blocks in order to get a continuous force transition, as shown in figure 2-6, see chapter 4. It is thus possible to manipulate the sharpness of the corner by specifying the curvature of the corner block. This makes for a fairly realistic corner model, since corners on real world solids have finite curvatures.
Chapter 3

Planes and Functions

So far the modeling of virtual surfaces has been limited to regular objects in 2-D and 3-D such as circles, spheres, squares, cubes, cones, etc. These types of objects have been fairly easy to model due to their symmetry and due to the fact that the walls of the squares and cubes all have been oriented parallel to the edges of the PHANToM workspace. The tests used to determine when the PHANToM point crosses a surface have therefore been fairly simple. New problems arise however, when one wants to model surfaces with a random orientation with respect to the workspace. In the following I will consider the modeling of randomly oriented planes and mathematical surfaces that are parallel to an axis of the workspace.

3.1 2-D Planes

The equation of a geometric plane in an orthogonal coordinate system is given by:

$$ax + by + cz + d = 0$$  \hspace{1cm} (3.1)

where $a, b$ and $c$ are the coordinates of the vector normal to the plane. We are interested in a procedure that can relate the position of the PHANToM point to the orientation of the plane. As a first step it would be useful to define a direction for the surface. The direction of a surface can be determined by defining a normal vector
to that surface. In absolute coordinates, any normal vector with a component in the positive z-direction is defined to be pointing in the positive direction of the surface. Thus, any point which lies at the tip of normal vector with a positive z-component can be said to lie above the plane surface. Similarly, any point which lies at the tip a normal vector with a negative z-component is said to lie below the surface.

Now that we have considered the issue of orientation of the PHANToM point, it is time to look at an algorithm that determines the actual proximity of the PHANToM point in relation to the plane surface. One can derive a formula for the distance from a point to a plane:

$$d = \frac{ax + by + cz}{\sqrt{a^2 + b^2 + c^2}}$$  \hspace{1cm} (3.2)

By having used the normal vector in its derivation, the above formula provides a quick and simple test for whether the point is above or below the plane: if the distance is positive, then the PHANToM point is above the plane, and if the distance is negative, the PHANToM point is below the plane. The force is then calculated the same way as before; by making it proportional to the depth of penetration of the PHANToM point.
3.1.1 Plane Segments

So far I have only dealt with equations for infinite planes. Infinite planes are not very useful when trying to construct solid objects of finite extent. In order to construct such objects one needs parts of planes.

A useful plane segment can be constructed by defining two limiting points. In the case of an infinite plane it is only necessary to check whether the PHANToM point is above or below the plane. It will however be different for a plane segment. Since the plane segment eventually will be used to represent a wall of a virtual object, force generation should only be considered when the projection of the PHANToM point onto the plane lies on the plane segment. In order to determine whether there should be a force generated it is not only necessary to check if the orientation of the PHANToM point, but also if the projection of the PHANToM point lies on the defined plane segment. The algorithm is given in the following pseudocode:

\[
\begin{align*}
\text{IF PHANToM point is below plane} & \quad \text{THEN} \quad <\text{project point onto plane}> \\
\quad & \quad \text{ELSE exit} \\
\text{IF projection is on plane segment} & \quad \text{THEN} \\
\quad & \quad <\text{calculate distance from PHANToM point to plane}> \\
\quad & \quad <\text{set force equal to stiffness times distance}> \\
\end{align*}
\]

This algorithm yields a block of semi-infinite thickness, and it is impossible to go around it; it is effectively a 1.5-D object. The entire projection region behind the plane segment is a force active area, and one must therefore beware of entering this region from the sides as this will suddenly suck the PHANToM point out through the surface (See figure 3-2).
3.2 Mathematical Functions

If mathematical functions could be used effectively as force reflecting surfaces, the repertoire of object shapes that could be created would be greatly increased. There are certain criteria that have to be met in choosing a function:

- The function has to be differentiable.
- The function has to be smooth.
- The function has to be continuous.

3.2.1 Analytical Solutions

The trick to creating a realistic force reflection model when dealing with mathematical functions is finding where the reaction force should act, and in what direction. Since the PHANToM point is allowed to penetrate the object, it can move around below the surface. It is thus possible for the operator to move parallel to a surface while being below it (See figure 3-3). The PHANToM point can therefore exit the object at a different spot from where it entered. This will not happen in real world
interactions, unless one is interacting with a fluid, in which case the force models would have to be changed. After experimentation with real world objects of varying compliance, I developed a model which generates a reaction force in the direction that minimizes the PHANToM point's path to free space. There are several ways to find the minimum distance from the interior PHANToM point to the surface of the object. The fastest is an analytical solution, which can be found can be found by differentiating the expression for the length of the vector from the PHANToM point to the curve, and setting it equal to zero (See figure 3-4). The correct root of this equation is the x-coordinate of the point on the curve which minimizes the distance.
to the PHANToM point. This distance is given by the following intrinsic equation:

\[ x + f(x) \frac{df(x)}{dx} - y_p \frac{df(x)}{dx} - x_p = 0 \]  

(3.3)

This analytical method is limited to continuous differentiable functions of degree lower than two, since higher orders will give equations of fifth degree and above, which are impossible to solve analytically. Functions of second order serve our purposes very well, however.

### 3.2.2 Numerical Solutions: Function Segments

![Figure 3-5: Numerical solution of shortest distance to a function segment.](image)

In order to treat general functions it is necessary to invoke numerical solutions. Generally, because of time and memory constraints, it is undesirable to employ numerical methods when dealing with virtual objects. Numerical methods can however be very effective if they use a limited number of iterations, even though this only yields approximate answers. An effective method seems to be a spin off of the bisection method. The method only has meaning if there are two starting points, i.e. when dealing with a function segment. The algorithm is given below in pseudocode:

BEGIN

counter = 0

END
FOR i=1 to i= <number of confining planes>

BEGIN

IF <PHANToM point is below plane i> THEN exit loop
ELSE increment counter by one
END FOR

IF count=i  THEN

IF <PHANToM point is below function (vertical distance!)> THEN

BEGIN

1. Calculate distance from PHANToM point to corners 1 and 2.
2. Find the line which bisects the angle between the lines from the PHANToM point to corner 1, and from the PHANToM point to corner 2.
3. Calculate the intersection between this line and the function.
4. Redo from 1, using the intersection point and whichever of points 1 and 2 that gave the shortest distance to the PHANToM point.

END

END IF

END IF

END IF

The procedure is illustrated in figure 3-5.
Chapter 4

Building Blocks

Semi-infinite blocks are a step in the right direction, but these blocks are still not very useful for creating solids of finite volume. It is therefore necessary to give the semi-infinite block a thickness. Assigning a thickness to the plane segment is a simple task and is done in the following pseudocode:

<create plane segments>
IF <PHANToM point penetration distance is greater than thickness>
   THEN <do not generate any force>
ENDIF

We have thus created a force reflecting region in space. It is very important to be aware that this is still a 1.5-D region. With an assigned thickness it is now possible to go around the object, but there is still only one force reflecting surface. Entry through the other surfaces will suck the PHANToM point through the object. The operator will only get the sensation of touching a wall when the PHANToM point enters from a certain direction, which is specified in the function call.

The problem with using the block discussed above when constructing complex arbitrary surfaces is its rectangularity. When putting these blocks together in order to make polygons, one gets corners with unrealistic force reflecting properties. This is due to the irregular region created by the overlap of the building blocks. It is therefore useful to consider a different type of building block.
4.1 Four Plane Blocks

![Diagram](image)

Figure 4-1: Building block defined by 4 planes.

An interesting possibility exists in employing overlapping planes. For example, several arbitrary planes all parallel to one of the workspace axes, will enclose a volume. This volume can form the basis for a virtual building block. After some experimentation I found that it was most effective to construct building blocks out of four plane segments. This would make the unification of blocks simpler when constructing larger 2-D objects. The block thus looks like the one presented in figure 4-1. Since this again is a 1.5-D block, the builder must specify which surface he wants to be force reflecting. The function was implemented such that this is done when the function is called. The algorithm for creating the block shown in figure 4-1 is given with the following pseudocode:

DEFINE <coordinates for the 4 corners of the block>
DEFINE <force reflecting surface, plane 1>
<calculate equation for force reflecting plane>

IF <PHANToM point is below the force reflecting plane>
    THEN <calculate equation for plane 2>
IF <PHANToM point is above plane 2>
THEN <calculate equation for plane 3>
IF <PHANToM point is above equation for plane 3>
THEN <calculate equation for plane 4>
IF <PHANToM point is above plane 4>
THEN <calculate distance from PHANToM point to plane 1>
<set force equal to stiffness times distance>
END IF

It is critical for the builder to have a clear picture of how he wants to assemble the building blocks he intends to use, because the orientation and the geometry of the blocks affect the conditions tested for. For example, one might want to test for the PHANToM point being below instead of above a plane for certain applications, and this needs to be specified in the function call through a control parameter.

4.2 Blocks with Function Surfaces

![Figure 4-2: Block with mathematical function as force reflecting surface.](image)

Blocks with linear force reflecting surfaces are of limited use. With them one can only create objects with straight edges and sharp corners. It would therefore be
interesting to make blocks that have smooth curved surfaces. This can be done by using the basic block construct discussed above with a modification. Instead of using a plane segment as the force reflecting surface one can use a mathematical function such as a polynomial expression or a transcendental function. An example of such a block is shown in figure 4-2.

A workable method for constructing such a block would be to define four points, and then connect these points with plane segments and a smooth differentiable continuous function.

4.3 The Odd-Parity Rule

![Diagram of Odd-Parity Rule](image)

Figure 4-3: Odd-Parity Rule for determining the interior of a polygon.

As mentioned, the speed of the computer puts constraints on the creation of virtual environments. The algorithms therefore have to be optimized. The problem when dealing with objects of random orientation is the time consuming procedure needed for checking whether the PHANTOM point is in the interior of an object. This problem has been dealt with in the field of computer graphics, and one of the most effective algorithms for determining whether a point is inside or outside a volume is called the Odd-Parity Rule [1]:

To determine whether a region lies inside or outside a given polygon, choose as a test point any point inside the particular region. Next, choose a ray that starts at the test point and extends infinitely in any direction,
and that does not pass through any vertices. If this ray intersects the polygon outline an odd number of times, the region is considered to be interior.

One way of implementing the Odd-Parity Rule is given in appendix A.1.
Chapter 5

Virtual Composite Objects

The preceding chapter discussed the creation of virtual building blocks. With these blocks it is now possible to create composite objects of arbitrary shape. Much like a child builds structures out of Lego blocks, we can construct virtual objects out of virtual Lego blocks.

5.1 Joining Blocks

![Diagram of a virtual composite object with a normal vector n.]

Figure 5-1: Testing for proximity of PHANToM point.

When many blocks are put together, the computer has to test for PHANToM
point interaction with every block belonging to the composite object. The number of tests the computer has to perform per servo loop is therefore greatly increased. This slows the servo rate down considerably. A way to reduce the number of tests is to define a circular region which circumscribes each block. This technique is a way of screening. For example, it is not necessary to check whether the PHANTOM point is in the interior of a block if it is not even within the circumscribing circle.

![Diagram of composite object](image-url)

Figure 5-2: Composite object.

When constructing polygons of arbitrary shape it is necessary for the builder to have a clear picture of how the building blocks should be assembled. The boundary points of the blocks have to be chosen carefully so the elements fit well together and thus produce a composite object that feels realistic. A master function which can generate a composite object given a mesh of points is currently being developed.
Chapter 6

Generalization to 3-D

The preceding chapters have dealt with objects that have been oriented parallel to one of the axes of the workspace. Thus, despite having depth, they are essentially 2-D objects. The next step is to generalize the current objects to 3-D, which is currently being done.

6.1 3-D Blocks

A 3-D haptic primitive is just a more complex version of the 2-D block primitive developed in chapter 4. Instead of using 3 or 4 points to define an area (actually a channel), we will now use 6 or 8 points to define a prismatic volume. The planes which form the facets of the blocks are calculated from these points, which are given in the function call.

Due to the complexity of the test, it is necessary to use a 3-D version of the Odd-Parity Rule described in chapter 4 to determine whether the PHANTOM point is in the interior of the block. Instead of constructing a ray originating in the PAHNTOM point, we will use a plane which is oriented parallel to one of the coordinate planes. If this plane fails to intersect the solid, then the PHANTOM point is outside. If the plane intersects the solid, it will produce a cross section, which is a mapping of the object onto 2-D space. Once this cross section is obtained, the test has degenerated to the 2-D test described in section 4-3. Before the test can be performed, the points
that span the cross section have to be computed. In order to do this one first has to find the equations for the lines which define the intersected edges of the object, and then find their intersections with the plane through the PHANTom point. The force is computed as before-proportional to the distance to the plane which the user has specified as force active, and in the direction of the plane’s normal vector.

The main problem with modeling 3-D solids of arbitrary shape is the large number of calculations that have to be performed each servo loop. Furthermore, the construction of composite objects in 3-D is going to require a master function which can fit blocks together in a highly complex geometry.
Chapter 7

Conclusion

It is clear that the efficiency of the software and hardware is a fundamental issue in the science of haptic environments. Many of the implementations discussed could be made considerably much faster and far more elaborate with proper program optimization. Efficient processing of constraints, is a key issue here. Furthermore, increased computing power would allow modeling of more complex environments.

7.1 Problems with the Vector Field Method

The methods discussed in the preceding chapters have all been vector field implementations, where the computer keeps a map of the force vectors at every location in the workspace. These force vectors are functions of the position and orientation of the points of interaction. When dynamic objects are included, the vector field becomes a function of the state of the environment as well. There is therefore a point where the vector field method becomes overwhelming, unless it is run on a mainframe computer.

Furthermore, since the Vector Field Method does not allow for a history, it is impossible to know the path taken by the PHANToM point in reaching a specific location. This particularly becomes a problem when dealing with thin objects. All haptic primitives have a maximum stiffness above which the haptic interface vibrates upon interaction. Therefore the PHANToM point must travel into the object before the force is large enough to make the object feel solid. If this distance is larger than
the thickness of the object, the vector field model breaks down.

7.2 The Future of Haptic Environments

A method addressing these problems is currently being researched. It is known as the Constraint Based God Object method[4]. This method provides a time history and permits the user to interact with virtual objects without penetrating them. Currently it seems that this method holds the most promise for the future development of haptic environments.

Furthermore, it is probable that many of the answers to our current problems can be found in the field of computer graphics, where the subject of modeling static and dynamic geometric environments has been treated in great depth.
Appendix A

Source Code (Turbo C)

A.1 Odd-Parity Rule Source Code

The Odd-Parity Rule is described in section 4.3. The coordinates of the corners of the block shown in figure 4-1 are passed as input parameters to the function. Also, a variable orientation is passed to the function to specify the orientation of the force reflecting surface.

```c
void odd_parity(float y1, float z1, /* Coordinates of building */
                float y2, float z2, /* block corners. */
                float y3, float z3,
                float y4, float z4,
                int orientation,
                float *forcex, float *forcey, float *forcez,
                float x, float y, float z)
{
    int coords[8]={y1, z1, y2, z2, y3, z3, y4, z4}
    int tempy, tempz, i, j=0;
    float z_coeff, y_coeff, distance;
    struct coordinates{int y; int z;};
    struct coordinates corners[4];
```
switch (orientation) {
    /* Finds distance between the PHANTOM point and the force reflecting plane. Also the y and z components of the forces are found.

    Note: distance = distance from PHANTOM point to force reflecting surface.

    y_coeff, 
    z_coeff = coefficients for directing the y and z forces. */

case 1: /* Force reflecting surface is the surface connecting points 1 and 2. */
    distance=(y*-(z2-z1)/(y2-y1))+z-(z1-y1*(z2-z1)/(y2-y1))/\
    sqrt(((z2-z1)/(y2-y1)*(z2-z1)/(y2-y1))+1.0);
    z_coeff=(y1-y2)/sqrt(((y2-y1)*(y2-y1)+(z2-z1)*(z2-z1));
    y_coeff=z_coeff*(z1-z2)/(y1-y2);
    break;

case 2: /* Force reflecting surface is the surface connecting points 2 and 3. */
    distance=(y*-(z3-z2)/(y3-y2))+z-(z2-y2*(z3-z2)/(y3-y2))/\
    sqrt(((z3-z2)/(y3-y2)*(z3-z2)/(y3-y2))+1.0);
    z_coeff=(y2-y3)/sqrt(((y3-y2)*(y3-y2)+(z3-z2)*(z3-z2));
    y_coeff=z_coeff*(z2-z3)/(y2-y3);
    break;

case 3: /* Force reflecting surface is the surface connecting points 2 and 3. */
    distance=(y*-(z4-z3)/(y4-y3))+z-(z3-y3*(z4-z3)/(y4-y3))/\
    sqrt(((z4-z3)/(y4-y3)*(z4-z3)/(y4-y3))+1.0);
    z_coeff=(y3-y4)/sqrt(((y4-y3)*(y4-y3)+(z4-z3)*(z4-z3));
    y_coeff=z_coeff*(z3-z4)/(y3-y4);
    break;
}
case 4: /* Force reflecting surface is the surface connecting points 2 and 3. */
  distance=(y*(-(z4-z1)/(y4-y1))+z-(z4-y4*(z4-z1)/(y4-y1)))/\
  sqrt(((z4-z1)/(y4-y1)*(z4-z1)/(y4-y1))+1.0);
  z_coeff=(y1-y4)/sqrt((y4-y1)*(y4-y1)+(z4-z1)*(z4-z1));
  y_coeff=z_coeff*(z1-z4)/(y1-y4);
  break;
  exit(1);
}

for(i=0;i<7;++2)
/* Storing the coordinates of the block corners (1,2,3,4), enabling for sorting. */
{
  coordinates.y=coords[i];
  coordinates.z=coords[i+1];
  corners[j]=coordinates;
  j++;
}

/* Bubble sorting coordinates according to z-values. */
for(i=0;i<3;++1)
  for(j=3;i<j;--j)
    if(corners[j-1].z<corners[j].z)
    {
      tempz=corners[j-1].z;
      tempy=corners[j-1].y;
      corners[j-1].z=corners[j].z;
      corners[j-1].y=corners[j].z;
      corners[j].z=tempz;
      corners[j].y=tempy;
if(z<corners[0].z && z>corners[3].y)
/* Checks if PHANToM point is between max(z) and min(z) */
{
    /* Define 3 regions between min(z) and max(z) */
    if(z>corners[1].z) region=1;
    else if(z>corners[2].z) region=2;
    else region=3;
    switch(region)
    {
    /* Each case represents a region. Applying odd-parity rule to witch ever region the PHANToM point might be in. */
    case 1: {
        /* Finding intersection between ray and surfaces of region */
        slope1=(corners[1].z-corners[0].z)/(corners[1].y-corners[0].y);
        z_intersect=corners[0].z-corners[0].y*slope1;
        y_cross1=(z-z_intersect)/slope1;
        slope2=(corners[2].z-corners[0].z)/(corners[2].y-corners[0].y);
        y_cross2=(z-z_intersect)/slope2;

        /* Testing if y-coordinate of PHANToM is bracketed by y-coordinates of the intersections. */
        if(y_cross1>y_cross2)
        {
            if(y_cross1>y && y>y_cross2)
            {
                /* Calculating reaction forces */
                *forcez+=z_coeff*distance*STIFFNESS;
                *forcey+=y_coeff*distance*STIFFNESS;
            }
        }
    }
else if(y_cross1<y && y<y_cross2)
{
    /* Calculating reaction forces. */
    *forcez+=z_coef*distance*STIFFNESS;
    *forcey+=y_coef*distance*STIFFNESS;
}
} break;

case 2: {
    slope1=(corners[1].z-corners[3].z)/(corners[1].y-corners[3].y);
    z_intersect1=corners[3].z-corners[3].y*slope1;
    y_cross1=(z-z_intersect1)/slope1;
    slope2=(corners[2].z-corners[0].z)/(corners[2].y-corners[0].y);
    z_intersect2=corners[2].z-corners[0].y*slope2;
    y_cross2=(z-z_intersect2)/slope2;

    if(y_cross1>y_cross2)
    {
      if(y_cross1>y && y>y_cross2)
      {
        *forcez+=z_coef*distance*STIFFNESS;
        *forcey+=y_coef*distance*STIFFNESS;
      }
    }
    else if(y_cross1<y && y<y_cross2)
    {
        *forcez+=z_coef*distance*STIFFNESS;
        *forcey+=y_coef*distance*STIFFNESS;
    }
} break;

case 3: {
    slope3=(corners[1].z-corners[3].z)/(corners[1].y-corners[3].y);
    z_intersect3=corners[3].z-corners[3].y*slope3;

37
y_cross1=(z_z_intersect3)/slope3;
slope2=(corners[2].z-corners[3].z)/(corners[2].y-corners[3].y);
z_intersect2=corners[2].z-corners[3].y*slope2;
y_cross2=(z-z_intersect2)/slope2;

if(y_cross1>y_cross2)
{
    if(y_cross1>y && y>y_cross2)
    {
        *forcez+=z_coeff*distance*STIFFNESS;
        *forcey+=y_coeff*distance*STIFFNESS;
    }
}
}
else if(y_cross1<y && y<y_cross2)
{
    *forcez+=z_coeff*distance*STIFFNESS;
    *forcey+=y_coeff*distance*STIFFNESS;
}
} break;

}
Bibliography


