Chih-Hao Ho[1]
Cagatay Basdogan[2]
Mandayam A. Srinivasan[3]
Laboratory for Human and Machine Haptics
Department of Mechanical Engineering and
Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, MA 02139
http://touchlab.mit.edu
[1]chihhao@mit.edu
[2]basdogan@mit.edu
[3]srini@mit.edu

Efficient Point-Based Rendering Techniques for Haptic Display of Virtual Objects

Abstract

Computer haptics, an emerging field of research that is analogous to computer graphics, is concerned with the generation and rendering of haptic virtual objects. In this paper, we propose an efficient haptic rendering method for displaying the feel of 3-D polyhedral objects in virtual environments (VEs). Using this method and a haptic interface device, the users can manually explore and feel the shape and surface details of virtual objects. The main component of our rendering method is the "neighborhood watch" algorithm that takes advantage of precomputed connectivity information for detecting collisions between the end effector of a force-reflecting robot and polyhedral objects in VEs. We use a hierarchical database, multithreading techniques, and efficient search procedures to reduce the computational time such that the haptic servo rate after the first contact is essentially independent of the number of polygons that represent the object. We also propose efficient methods for displaying surface properties of objects such as haptic texture and friction. Our haptic-texturing techniques and friction model can add surface details onto convex or concave 3-D polygonal surfaces. These haptic-rendering techniques can be extended to display dynamics of rigid and deformable objects.

1 Introduction

Computer graphics—concerned with the generation and rendering of graphical images—has established itself as a major area of research in the past few decades. Recent developments in force-reflecting devices that enable the user to touch, feel, and manipulate virtual objects have given rise to a need for the systematic development of methods to generate and display virtual objects with haptic attributes. We have proposed the term *computer haptics* to encompass the research concerned with the generation and haptic rendering of virtual objects (Srinivasan & Basdogan, 1997).

The goal of haptic rendering is to display the haptic attributes of surface and material properties of virtual objects in real time via a haptic interface device. Haptic display of 3-D objects in VEs, with various applications in many areas of human-machine interactions, has emerged as an exciting and challenging research topic for scientists and engineers during the last few years. (Refer to Srinivasan (1995) and Burdea (1996) for an overview of hardware, human factors, and applications; Srinivasan and Basdogan (1997) for an overview of our work, including haptic rendering; and Salisbury et al. (1995) and Salisbury and Srinivasan (1997) for Phantom-based haptics.) It is generally accepted that, for

Presence, Vol 8, No 5, October 1999, 477–491

the "touch and feel" of objects in VEs to appear natural, the haptic servo rate should be of the order of 1,000 Hz. In other words, the computational time for each haptic servo loop (where a reaction force vector is computed for each position input) should take about a millisecond or less. If the 3-D objects in VEs are simple primitives (e.g., cube, cone, cylinder, sphere, etc.), this goal can be easily achieved. However, realistic synthetic environments usually require multiple 3-D objects that have complex surface as well as material properties. Therefore, the development of efficient haptic-interaction techniques that can render arbitrary convex and concave 3-D objects in a time-critical manner becomes essential. We view the haptic rendering of virtual objects to consist of two parts: (1) a *haptic-interaction paradigm* that defines the nature of the "haptic cursor" and its interaction with object surfaces, and (2), an *object property display algorithm* to render surface and material properties of objects through the repeated use of the haptic-interaction paradigm.

Initial haptic-rendering methods focused on displaying simple object shapes. Massie and Salisbury (1994) developed the PHANToM haptic interface device and proposed a simple method for the haptic rendering of objects. In this haptic-interaction model, the user could interact with primitive 3-D objects in VEs through the end point of the haptic device, which is defined as the haptic interface point (HIP). Later, Zilles and Salisbury (1995) developed a more sophisticated, constraint-based method to render generic polygonal meshes. They defined a "god-object" point to represent the location of a point that is constrained to remain on a particular facet of the object. Lagrange multipliers are used to compute the new location of the god-object point such that the distance between the god-object and the haptic interface point is minimized. Adachi et al. (1995) and Mark et al. (1996) suggested an intermediate representation (a tangent plane) to render virtual surfaces. Although the forces are updated frequently (~1 kHz), the tangent plane is updated more slowly. Ruspini et al. (1997) proposed an approach similar to the god-object technique and improved the collision-detection algorithm by constructing a bounding sphere hierarchy and configuration space. In a departure from the point-based methods described above, Basdogan et al. (1997) proposed a ray-based interaction technique in which the probe is modeled as a line segment rather than a point. Using the ray-based interaction technique, we can simulate the contact between the tip as well as the side of the probe with several convex objects at the same time. We can then compute the associated forces and torques to be displayed to the user.

In addition to the interaction paradigms described above, additional techniques have been proposed for displaying surface properties such as shape, friction, and texture of virtual objects. Morgenbesser and Srinivasan (1996) have proposed force-shading methods to smooth the feel of polyhedral objects by eliminating the force discontinuities at polygonal boundaries. Salcudean and Vlaar (1994), Salisbury et al. (1995), Chen et al. (1997) and Green and Salisbury (1997) have proposed techniques to simulate friction on smooth surfaces. Minsky et al. (1990, 1995) have proposed algorithms to simulate textures on 2-D surfaces. Siira and Pai (1996) and Fritz and Barner (1996) have also proposed methods to generate stochastic textures on well-defined surfaces. Basdogan et al. (1997) have developed techniques for the haptic texturing of polyhedral objects.

In this paper, we describe an efficient haptic-interaction paradigm as well as algorithms for the haptic display of surface properties. One of the major concerns in haptic interaction is that the computational time usually increases with the number of polygons. In such a case, the quality of the haptic interaction will depend on the complexity of the environment. If the reflected forces cannot be updated at a sufficiently high rate, contact instabilities occur. In order to have stable haptic interactions, an efficient collision-detection algorithm that makes the computational time essentially independent of the number of polygons of an object would be quite beneficial (Ho et al., 1997). We present a haptic-interaction method that utilizes a hierarchical database, a client-server model, and a local search technique geared towards achieving this goal. To display object surface properties, we extend the friction model proposed by Salcudean and Vlaar (1994) and Salisbury et al. (1995) to simulate bumpy frictional surfaces by spatial variations in the values of static and dynamic friction coefficients. We also present a

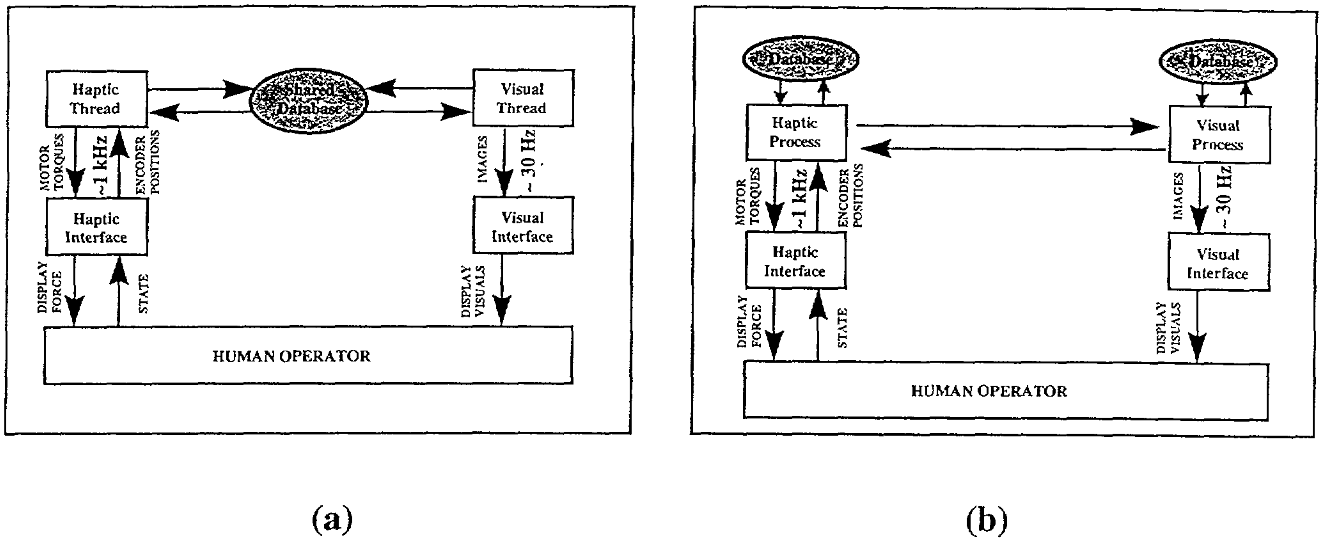

Figure 1. *Software architectures for (a) multithreading and (b) multiprocessing. In the multithreading structure, both the haptic and the graphic loops share the same database. In this structure, the synchronization of the two loops in accessing the data is important. In the multiprocessing structure, the haptic and the graphic loops have their own copies of databases that are not shared. The two processes could run on the same machine or on different machines. The communication protocol between the two loops that ensures consistent update of both graphics and haptics databases is important in this structure.*

technique that can generate textures on arbitrary 3-D surfaces. This technique is similar to the one proposed by Basdogan et al. (1997), but it has been extended to simulate textures with a larger range of height and spatial frequency.

In the next section, we first discuss the client-server models for synchronizing haptic and visual modalities and displaying them in an efficient manner. Our collision-detection algorithm and the hierarchical data structure for storing geometrical properties of polyhedral objects are described in Section 3. In Section 4, we present several methods for displaying surface properties of objects in VEs via a haptic interface. The results and conclusions of the study are summarized in Section 5 and 6, respectively.

2 Software Architecture for the Integration of Haptics and Graphics

The hardware components of our setup include a computer running the Open Inventor graphics toolkit to display the graphical model of the 3-D virtual environment, and a force feedback device, PHANToM (SensAble Technologies, Inc.), to convey to the user a sense of touch and feel of virtual objects in this environment. To have a satisfying experience in interacting with a VE, the graphics and haptic update rates need to be maintained at around 30 Hz and 1,000 Hz, respectively. In order to develop effective multimodal VEs and the optimal usage of the CPU capabilities, we have experimented with multithreading and multiprocessing techniques and successfully synchronize the visual and haptic servo loops (Ho et al., 1997). In both models, the haptics and graphics processes are server and client, respectively. The conceptual difference between the multithreading and the multiprocessing structures is illustrated in Figure 1.

Our experience is that both multithreading and multiprocessing techniques are quite useful in achieving high graphic and haptic rendering rates and stable haptic interactions. The choice of multiprocessing or multithreading structures should depend on the characteristics of the application. Although creating a separate process for each modality seems to require more programming effort than multithreading, it enables the user to display the graphics and/or haptics on any desired machine(s). If large amounts of data will need to be transferred back and forth frequently between the loops, we recommend multithreading techniques implemented with timer callbacks for synchronizing the haptic and visual servo loops.

3 Haptic-Interaction Paradigm

Two important issues any haptic-interaction paradigm has to specify are (1) *collision detection:* the detection of collisions between the end point of the generic probe and the objects in the scene, and (2) *collision response:* the response to the detection of collision in terms of how the forces reflected to the user are computed. A good collision-detection algorithm not only reduces the computational time, but also helps in correctly displaying interaction forces to the human operator to make the haptic sensing of virtual objects more realistic. Whereas the collision-detection procedures for haptics and graphics are the same, the collision response in haptic rendering differs from that in computer graphics. In computer graphics, collision-detection (Cohen et al., 1995; Lin, 1993; Gottschalk et al., 1996; Smith et al., 1995; Hubbard, 1995) techniques are used to detect if two objects overlap. When the collision is detected, objects are separated from each other using collision-response methods (Moore & Wilhelms, 1988; Baraff, 1994; Mirtich, 1996; Mirtich & Canny, 1995). In general, the purpose of the collision detection and response in graphics is to avoid the overlap between objects and to simulate the behavior of objects following the overlap.

In contrast, the purpose of collision detection in haptic rendering is not only to check collisions between objects, but more frequently to check collisions between the probe and virtual objects to compute the interaction forces. When simulating interactions between the probe and the objects, the reflected force typically increases with the penetration distance such that it resists the probe from further penetrating the object. Thus, the end point of the probe will always be inside the object during the collision-response phase. This is a major difference between the collision-response techniques developed for haptic and graphic interactions. In graphics, typically the existence of the overlap needs to be detected, followed by collision-response algorithms. In haptics, the main goal is to compute the reaction force. Hence, the depth of penetration and how the penetration evolves are important. After detecting a collision between the simulated probe and the objects, a simple mechanistic model—such as Hooke's law ($F = kx$, where x is the penetration vector)—can be used to calculate the force. One simple way to determine the depth of penetration is to use the shortest distance between the probe tip and the object's surface (Massie, 1993; Massie & Salisbury, 1994). This approach works well for primitive objects such as a cube, sphere, cylinder, etc; however, the drawbacks of this technique are that it cannot display the objects that are thin or polyhedral (Massie, 1993; Ruspini et al., 1996, 1997). Another approach to decide the depth of penetration is to use a constraint-based method (Zilles & Salisbury, 1995). This method defines an imaginary point—the god-object point—that is constrained by the facets of the object. The penetration depth could then be defined as the distance from the god-object to the probe tip. This approach can be applied to polyhedral objects, even when the objects are thin. However, this method requires different sets of rules to handle concave and convex objects.

In the following sections, we propose "neighborhood watch," an efficient haptic-rendering technique that can handle both convex and concave objects uniformly. Although this algorithm is conceptually similar to the god-object algorithm (Zilles & Salisbury, 1995), it follows a different approach. We believe that our approach reduces the computational time, makes the haptic servo rate independent of the number of polygons of the object, and results in more-stable haptic interactions with complex objects.

3.1 Hierarchical Database

In our approach, the first step is to construct a hierarchical database to store the geometrical properties of the 3-D objects. We use the Open Inventor file format for graphical display of 3-D objects in VEs. After loading

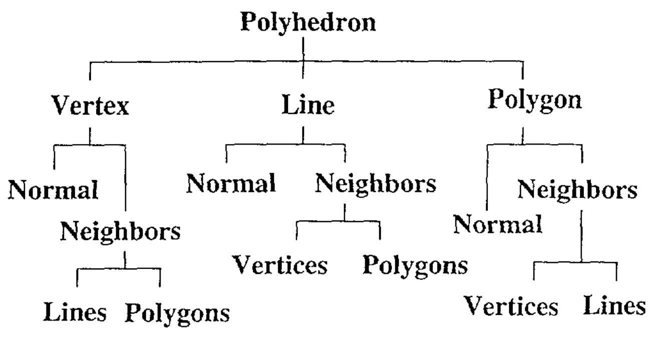

Figure 2. *The hierarchical database. The polyhedron representing the object is composed of three primitives: vertex, line, polygon. Each primitive is associated with a normal vector and a list of its neighbors*

the Inventor object files (3-D objects are assumed to be made of indexed triangles), the coordinates of the vertices and the indices of the polygons are automatically stored in our hierarchical database. Using this information, we construct another type of 2-D geometrical primitive—namely, the lines that are simply the edges of the triangular polygons. As a result, the polyhedral objects in our own database are made of three types of 2-D geometrical primitives: polygons, lines, and vertices. In order to implement a fast search technique for detecting collisions between the probe and 3-D objects, we extend our database such that each primitive has a list of its neighboring primitives. Each polygon has neighboring primitives of lines and vertices; each line has neighboring primitives of polygons and vertices; and each vertex has neighboring primitives of polygons and lines (Figures 2 and 3). In addition to the connectivity information, we compute the normal vector of each primitive. For a polygon, the normal is the vector that is perpendicular to its surface and points outwards. For a line, its normal is the average of the normals of its neighboring polygons. For a vertex, its normal is the average of the normals of its neighboring polygons, linearly weighted by the associated angle subtended at the vertex.

3.2 Collision Detection and Response

When exploring virtual environments, we interact with objects through the end point of the probe (the HIP). At the same time, we also consider another point—the Ideal Haptic Interface Point (IHIP), which is similar to the god-object point (Zilles & Salisbury, 1995)—to follow the trace of the HIP. The HIP is not constrained and, consequently, can penetrate the object. However, we constrain the IHIP such that it does not penetrate any object. When the HIP is outside the virtual object, the IHIP will be coincident with the HIP. If the HIP penetrates a virtual object, the IHIP will stay on the surface of the object. When the HIP is outside the object, we keep track of its path and check if this path penetrates any polygon. For this purpose, we construct a line segment between the previous and current coordinates of the HIP and detect the collisions between this line segment and the polygons of the 3-D object. (Because the servo rate is around 1 kHz and human motions are relatively slow, this line is very short.) To achieve fast detection of collisions between the line segment and the polygonal objects, we utilize the "hierarchical bounding boxes" approach (Gottschalk et al., 1996). In this approach, polygons of the 3-D object are hierarchically partitioned a priori until each polygon is enclosed by its own bounding box. The detection of collisions occurs in three consecutive stages. First, collisions of the line segment joining the current and previous HIP with the bounding box of the object are detected. If the line segment is inside the bounding box, then collisions are checked with the partitioned bounding boxes along the branches of the hierarchical tree by marching from the top of the tree downwards. As collisions are detected with successive bounding boxes along a particular branch of the tree, the last collision is checked between the line segment and a polygon itself at the lowest level of the tree. If the line segment penetrates a polygon, we set this polygon as the contacted geometric primitive. The IHIP will then be constrained to stay on the surface of this polygon. The nearest point from this polygon to the current HIP is set as the IHIP, and the distance from the IHIP to the current HIP is set as the depth of penetration.

Although the first contacted geometric primitive is always a polygon and the IHIP is assigned to be on the surface of this polygon, it can easily be a line or a vertex in subsequent iterations (i.e., the IHIP can be constrained to stay on the edge or vertex of a polygon as

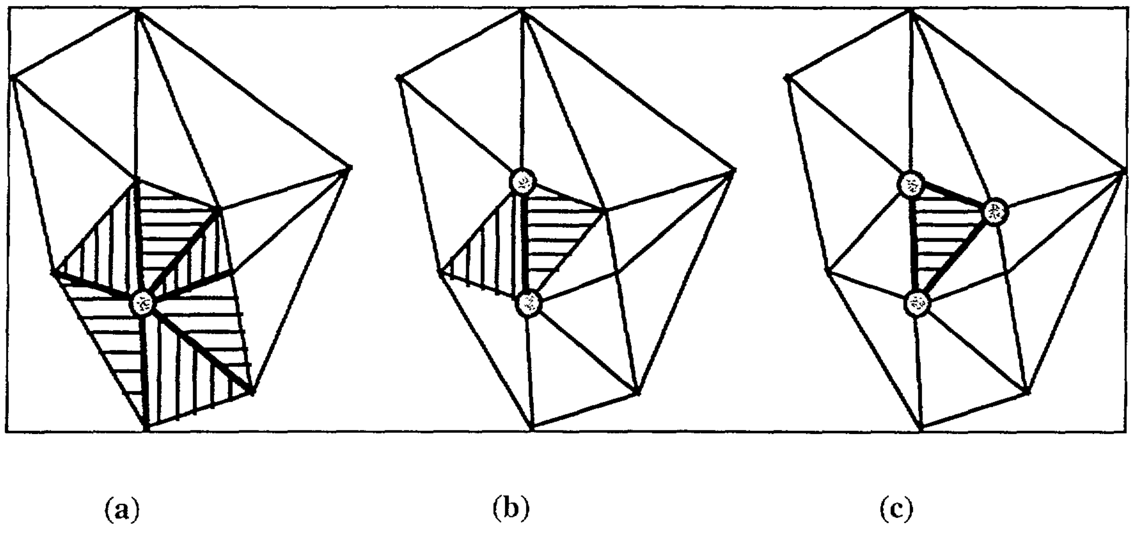

Figure 3. *Illustration of how to define the neighbors of a vertex, line, and polygon for an object. In this figure (a), the vertex has six neighboring lines and six neighboring polygons. (b) the line has two neighboring vertices and two neighboring polygons. (c) the polygon has three neighboring vertices and three neighboring lines.*

well as on the surface of the polygon). In the next iteration, we calculate the nearest distances from the current HIP to the contacted geometric primitive and its neighboring primitives. For example, if the contacted primitive is a polygon, then we check the distance from the current HIP to the neighboring lines and vertices. This local-search approach significantly reduces the number of computations and also makes them essentially independent of the number of polygons that represent the objects. Then, we set the primitive that has the shortest distance to the current HIP as the new contacted geometric primitive and move the IHIP to a point that is located on this primitive and is nearest to the current HIP. (See Figure 4.) This rule-based algorithm is repeatedly applied for the ensuing interactions.

In each cycle, one also needs to check if the current HIP is still inside the virtual object. For this purpose, we construct a vector from the current HIP to the IHIP. If the dot product of this vector and the normal of the contacted primitive is negative, the current HIP is no longer inside the object and there is no longer any penetration. If the dot product is positive, then the current HIP is still inside the object, and the magnitude and direction of the vector from the current HIP to the IHIP can be used for the force computations. The pseudocode for our haptic-interaction algorithm we call "neighborhood watch" is given below (Ho et al., 1997).

```
if (collision == FALSE)
{
 if (the path of HIP penetrates a polygon)
 {
  Set the polygon as the contacted geometric
  primitive
  Move the IHIP to a point on this surface that is
  closest to current HIP collision ← TRUE;
 }
}
else
{
 contacted geometric primitive ← the contacted
 geometric primitive in the previous loop;
 primitive1 ← contacted geometric primitive ;
 distance1 ← closest distance from current HIP to
 primitive1;
 repeat {
  primitive1 ← contacted geometric primitive;
  for (i=1 : number of neighboring primitives of
  primitive1)
  {
```

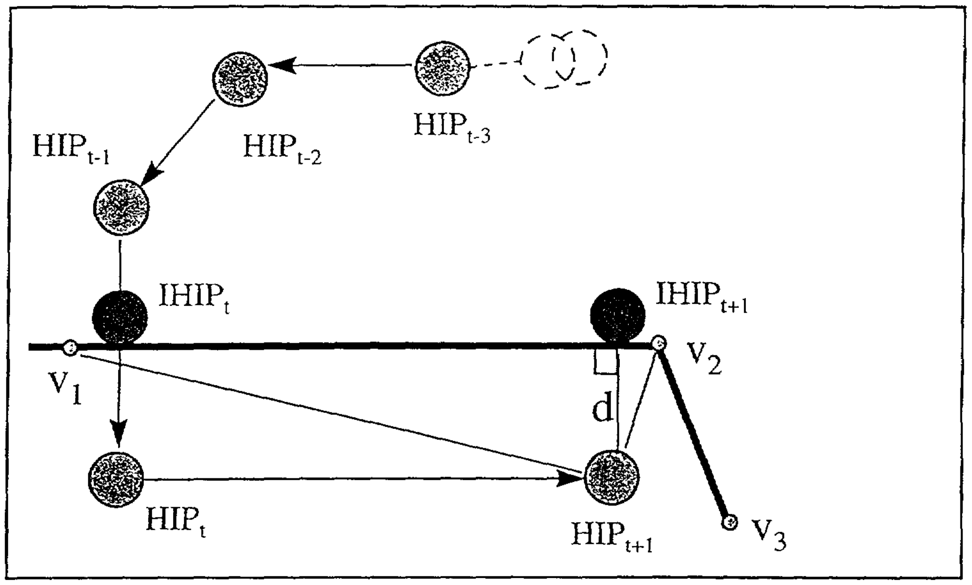

Figure 4. *Haptic interactions between the end point of the probe and 3-D objects in VEs: Before the collision occurs, the HIP is outside the object surface and is identical with the IHIP (see HIP$_{t-3}$, HIP$_{t-2}$, and HIP$_{t-1}$). When the HIP penetrates into object at time = t, the IHIP is constrained to stay on the surface. At time = t + 1, HIP moves to a new location (HIP$_{t+1}$), and the new location of IHIP is determined from the current HIP and its neighboring primitives based on the nearest distance criterion.*

```
      primitive2 ← the ith neighboring primitive of
      primitive1;
      distance2 ← distance from current HIP to
      primitive2;
      if (distance2 < distance1)
      {
        contacted geometric primitive ← primitive2;
        distance1 ← distance2;
      }
    }
  } while (primitive1!= contacted geometric
    primitive)
  Move IHIP to a point that is nearest from the
  contacted geometric primitive to current HIP
  vector1 ← vector from current HIP to current IHIP;
  normal1 ← normal of the contacted geometric
  primitive;
  if (dot product (vector1, normal1) <0)
    collision ← FALSE;
  else
  {
    collision ← TRUE;
    penetration vector ← vector1;
    penetration depth ← magnitude of penetration
    vector;
  }
}
```

Using this algorithm, we can haptically render both convex and concave objects in an efficient manner. The computational time for detecting the first collision will be of the order of log(N) for a single object, where N is the number of polygons (Gottschalk et al., 1996). After the first collision, we consider only the distances from the current HIP to the contacted primitive and its neighbors to determine the new location of IHIP; therefore, the servo rate will be fast because it depends only on the number of the contacted geometric primitive's neighbors. For a homogeneously tessellated polyhedron, as N increases, because the number of computational operations for searching the neighbors of each primitive is about the same, the servo rate will continue to be independent of the total number of polygons after the first collision.

4 Haptic Display of Surface Properties

When we explore objects in the real world, we seldom contact smooth, frictionless surfaces. Therefore, in addition to the shape of objects, surface property is another important factor that could increase the naturalness of VEs. In our simulations, surface properties are separated into two different categories: texture and friction. From a computational viewpoint, texture is expensive to represent as shape. However, both friction and texture can be simulated by appropriate perturbations of the reflected forces. The major difference between the friction and the texture simulation via a haptic device is that the friction model creates only lateral forces and the texture model modifies both the lateral and normal forces.

The simulation of static and dynamic friction gives users the feeling as if they are stroking the surface of sandpaper (Salisbury et al., 1995). By changing the mean value of the friction coefficient and its variation, we can efficiently simulate various surfaces (Siira & Pai, 1996; Green & Salisbury, 1997). Textures can be simulated by simply mapping bumps and cavities onto the surface of objects. Minsky (1995) presented a method to simulate 2-D haptic textures by perturbing the direction of reaction force. In contrast to the simulation of surface properties that add roughness to a smooth surface, the force-shading technique (Morgenbesser & Srinivasan, 1996; Basdogan et al., 1997; Ruspini et al., 1997) eliminates the surface roughness. When the objects are represented as polyhedrons, the surface normal and the force magnitude are usually discontinuous at the edges. By using the force-shading technique, we can reduce the force discontinuities and make the edges of polyhedral object feel smoother.

4.1 Force Shading

During haptic rendering, we can compute the point of contact and retrieve information about the contacted primitive from the database. To compute the shaded force vector, we consider only the vertex normals of the original unshaded surface. In this regard, if the contacted primitive is a vertex, the normal of the vertex

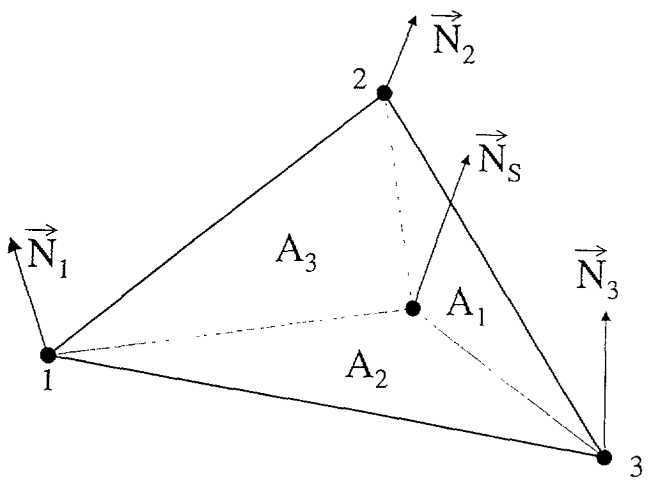

Figure 5. *The collision point (IHIP) divides the collided triangle into three subtriangles. The point in the center is the collision point. Points 1, 2, and 3 are the three vertices of the triangle. $\vec{N}_i$'s are the normals of the vertices, A_i's are the areas of subtriangles, and $\vec{N}_S$ is the normal at the collision point.*

is used directly as the normal of the collision point. If the contacted primitive is a line, the normal at the collision point is the average of the normal of the line's two neighboring vertices, weighted by the inverse of the distance from the collision point to the vertices. If the contacted primitive is a polygon, because our objects are made of triangles, the collision point will divide the collided triangle into three subtriangles. (See Figure 5.) We then calculate the normal ($\vec{N}_S$) at the collision point by averaging the normals of the vertices ($\vec{N}_i$) of the contacted polygon, weighted by the areas A_i of the three subtriangles (Equation 1).

$$\vec{N}_S = \frac{\sum_{i=1}^{3} A_i \cdot \vec{N}_i}{\sum_{i=1}^{3} A_i} \tag{1}$$

Although the interpolation of the vertex normals makes the normal vector $\vec{N}_S$ continuous at the edges, the user can still feel the existence of the edges due to the discontinuities in the force magnitude. (Because of the limited position-tracking ability of the user, the depth of penetration is not held constant while the operator moves the probe of a haptic device from the surface of one

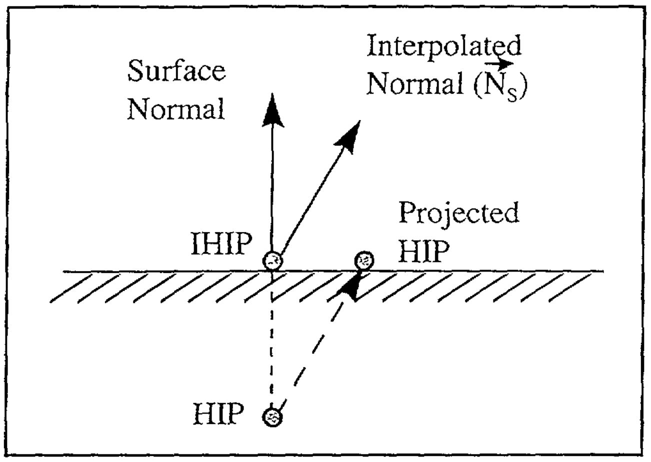

Figure 6. *The direction and magnitude of force with force-shading technique. Before applying force shading, the vector from HIP to IHIP is used to determine the force. After applying the force-shading technique, the vector from HIP to the projected HIP is used to compute the force.*

polygon to another to explore the shape of a polyhedron.) To minimize this problem, we project the HIP to the object surface along the direction of the interpolated normal vector ($\vec{N}_S$). The distance between the projected HIP and the current HIP is then used to determine the magnitude of the force. (See Figure 6.) This method works well when the penetration of HIP is small compared to the size of the polygon. If the penetration depth is larger than the length of the edge of the contacted polygon, the users can still feel the existence of the edges. Force shading should be implemented appropriately depending on the geometry of the object that is being represented. It should be turned on to minimize the effects of artificial edges created by the polygonal representation, whereas it should be turned off when the objects have natural edges (e.g., the force-shading technique should be used to smooth the lateral surfaces of a circular cylinder made of polygons and not for the flat ends.)

4.2 Friction

As mentioned earlier, friction can be simulated by adding a lateral force vector to the normal force vector. We extended the friction model proposed by Salcudean and Vlaar (1994) and Salisbury et al. (1995) to simulate various types of bumpy frictional surfaces by changing the static and dynamic friction coefficients at different spatial locations and to improve the perception of object surfaces. We can simulate static and dynamic friction such that the user feels the stick-slip phenomenon when he/she strokes the stylus of a haptic device over the object surface.

Our model has two types of friction states: sticking and sliding. During the sticking and sliding states, the static and dynamic friction coefficients are used, respectively. When the first collision is detected, the contacted point is stored as the sticking point and the friction state is set as the sticking state. When the users move the HIP, a tangential resistive force is applied to the user. The magnitude of the tangential force is decided by the linear elastic law ($\vec{F}_s = k_s\vec{x}$, where k_s is proportional to the static friction coefficient, and $\vec{x}$ is the vector from the IHIP to the sticking point). If the tangential force is larger than the allowable static friction force (the normal force times the static friction coefficient), the friction state is changed to the sliding state and the sticking point is updated and moved to a new location. The location of the new sticking point is on the line between the old sticking point and the IHIP. The distance from the IHIP to the new sticking point is calculated using the inverse of linear elastic law ($\vec{x} = \vec{F}_d/k_d$, where k_d is proportional to the dynamic friction coefficient, $\vec{F}_d$ is the friction force that is equal to the normal force times the dynamic friction coefficient). It should be noted that k_d is smaller than k_s, as is the case for real objects. Unlike the method described in Salisbury et al. (1995), we keep the state in the sliding state, instead of switching to the sticking state, following the movement of sticking point. The reason is that, if we change back to the sticking state right after moving the sticking point, we observed that the feeling of dynamic friction is lost. In the next iteration, we calculate the tangential force ($\vec{F}_d = k_d\vec{x}$). If this force is larger than the dynamic friction force (the normal force times the dynamic friction coefficient), we know that the user intends to keep moving the HIP in the same direction, and, therefore, we keep the state in the sliding state and use the method mentioned above to move the sticking point to a new position. However, if the force (F_d) is smaller than the dynamic friction force (meaning that the user has stopped moving in the same direction), we change the state to sticking and do not update the position of the sticking point. The computation of frictional force is done continuously as long as the HIP is inside the object.

Uniform friction all over the object surface is created when the static and dynamic friction coefficients are constant and independent of the position. We can also create periodic frictional surfaces by varying the static and dynamic friction coefficients at different locations. More-sophisticated surfaces can be simulated by changing the distribution of the friction coefficients. Green and Salisbury (1997) have shown that various grades of sandpaper can be simulated well by modifying the mean and variance of the friction coefficient.

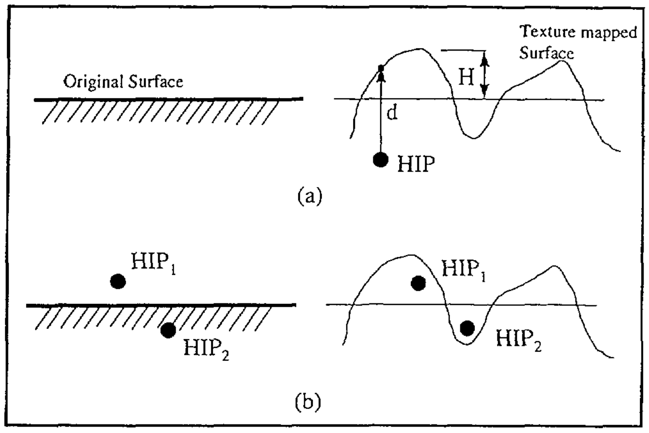

Figure 7. *Collision situations may change after texture mapping. The original surface and the surface after texture mapping are shown in (a). In (b), there is no collision between HIP$_1$ and the original surface before the texture is mapped, but collision occurs after the texture mapping. The situation is the opposite for HIP$_2$.*

4.3 Texture

Texturing techniques can be used to add roughness to a smooth surface. Similar to the role of textures in computer graphics, haptic textures can add complexity and realism to the existing geometry. Texturing techniques reduce the load on the geometry pipeline because they reduce the need for expressing texture geometry explicitly. In computer graphics, the final goal of the texturing computations is to decide the color in each pixel. Similarly, the final goal in haptic texturing is to decide the direction and magnitude of the force that will be reflected to the user. In order to simulate haptic textures, we need to know IHIP, HIP, and the height field that will be mapped to the surface. How to compute IHIP and HIP has already been described in Section 3, and how to map the height field over the object surfaces to simulate textures will be described in this section.

4.3.1 Magnitude of Force. When there is no texture on the surface, we use the elastic law to decide the magnitude of the force based on the depth of penetration. When a height field is mapped over the surface to simulate haptic textures, the geometry of the surface will be changed, which, in turn, will change the depth of penetration. To correctly change the force, we simply need to add the texture height at IHIP to the depth of penetration. We choose the height value at IHIP instead of HIP, because IHIP is the simulated contact point and is the point that stays on the surface of the object. If we use the height value at HIP, different penetration depths will have different height values, although the contact point is the same.

One other situation that has to be taken into account in texture display is the collision state. Adding a texture field over a surface may change the collision state from no collision to collision or vice versa when the probe is moved over the textured surface (Figure 7). There may be a collision between the HIP and the texture-mapped surface even when no collision exists between the HIP and the original surface.

To handle all such cases, we compute the nearest distance to HIP at each iteration using the following rules. If there is a collision with the original surface, the nearest point to the HIP is IHIP, and the nearest distance is the distance between the IHIP and HIP. Because the HIP is below the contacted geometric primitive, we consider its value as negative. If there is no collision between the HIP and the primitive, the nearest distance is the distance between the HIP (the HIP is above the surface in this case), and the primitive and its value is positive. With the definition of the nearest-distance concept, we can easily check the collision status: the collision with textured surface actually happens only when the nearest distance is less than the height value at the nearest point on the original surface. (Note that the height value could be positive or negative relative to the original object surface.) We can define the depth of penetration to be the height value at the nearest point minus the nearest distance. If this depth of penetration is negative, it is set to zero. The magnitude of the force is then calculated based on the depth of penetration and the mechanistic law that governs the interactions.

4.3.2 Direction of Force. To decide the direction of the force in the haptic display of texture, we modify the "bump-mapping" technique of computer graphics. Bump mapping (Blinn, 1978) is a well-known graphical technique for generating the appearance of a non-smooth surface by perturbing the surface normals. In haptics, if we perturb the direction of the force, we can also generate a similar effect that makes the users feel that there are bumps on the smooth surface. The first step in deciding the direction of force is to determine the direction of the surface normal. For graphics, Max and Becker (1994) improved the original bump-mapping technique and suggested a direct method of mapping that does not require a transformation from global to parametric space. They developed a formulation that is purely based on the original surface normal and the local gradient of the height field that generates bumps on the surface of the object. Max and Becker utilized this technique to generate bumps to graphically simulate clouds. We used the same approach to calculate the perturbed surface normals ($\vec{M}$).

$$\vec{M} = \vec{N}_s - \vec{\nabla h} + (\vec{\nabla h}.\vec{N}_s)\vec{N}_s \tag{2}$$

$$\vec{\nabla h} = \frac{\partial h}{\partial x}\hat{i} + \frac{\partial h}{\partial y}\hat{j} + \frac{\partial h}{\partial z}\hat{k} \tag{3}$$

where $\vec{M}$ represents the normal of the surface after texture mapping, $h(x, y, z)$ represents the height (texture) field function, $\vec{\nabla h}$ is the local gradient vector, and $\vec{N}_s$ represents the unperturbed surface normal at the collision point.

The most intuitive way of deciding the direction of the force is to use the perturbed surface normal ($\vec{M}$) as the force direction. Indeed, our experience shows that this can give users a very good sensation of texture in most cases. However, if the amplitude and spatial frequency of the texture are very high and the force applied by the user to explore the surface is very large, the haptic device will become unstable due to the sudden changes in the force magnitude and direction when this algorithm is used. The reason for the instability is that the magnitude and direction of the force change abruptly with only a small change in position. To eliminate the force instability, we modify the approach slightly and use the normal of the original surface along with the perturbed surface normal to calculate the direction.

$$\vec{F} = (d - Kh)\vec{N}_s + Kh\vec{M} \quad \text{if } d \geq \text{Kh} \tag{4a}$$

$$\vec{F} = d\vec{M} \quad \text{if } d < \text{Kh} \tag{4b}$$

where $\vec{F}$ represents the force that will be displayed to the users, d represents the magnitude of the penetration, $\vec{N}_s$ represents the normal of the original surface, $\vec{M}$ represents the perturbed surface normal, K is a scalar that depends on the properties of the texture, and h is the height of the texture. From Equation (4a) and (4b), we observe that the force will be in the same direction as the perturbed surface normal $\vec{M}$ when the penetration is small (i.e., the magnitude of the force is small) compared to the height of the texture. This is the same as in the bump-mapping approach. However, if the force is large, only a small amount (the Kh term in Equation (4a)) of the force is perturbed, and the remaining amount will still be along the original surface normal. This modification will make the interactions with fine textures more stable. (In our experience with PHANToM force feedback device, the simulation gives the best results when the value of K is between 1 and 2. If the simulated texture is very smooth, K should be set to 2. If the changes in texture gradient are sharp, then K should be set to 1 or even smaller).

4.3.3 Height Field of Textures. In order to apply the proposed texturing techniques, textures must be C^0 and C^1 continuous (Foley et al., 1995). The simulation has the best effect if the height and the wavelength (i.e., the inverse of the spatial frequency) of the texture are of the same order. Taking these constraints into account, we have ported several texturing techniques of computer graphics to simulate haptic textures. The haptic texturing techniques can be classified into two parts: (a) image-based and (b) procedural.

(a) Image-based haptic texturing: This class of haptic texturing deals with constructing a texture field from 2-D image data. In computer graphics, the digital images are wrapped around 3-D objects to make them look more realistic. The graphical texture map consists of texels with only 2-D color or grayscale intensities, whereas the haptic texture map consists of texels with a height value (Basdogan et al., 1997).

The first step to create image-based haptic texture is to map the digitized image to the 3-D polygonal object. We use the two-stage texture-mapping techniques of computer graphics (Bier & Sloan, 1986; Watt & Watt, 1992) to map the 2-D image to the surface of 3-D objects. The first stage is to map the 2-D image to a simple intermediate surface such as plane, cube, cylinder, or sphere. The second stage maps the texture from the intermediate surface to the object surface. After this two-stage mapping, we can obtain the height value for any point on the object surface (See Basdogan et al. (1997) for implementation details.)

After the mapping, the only information we need to create the haptic texture is the gradient of the height field at the IHIP. The gradient of the height could be computed using the central difference approximation for partial derivatives:

$$\frac{\partial h}{\partial x} = \frac{(h_{x+\epsilon} - h_{x-\epsilon})}{2\epsilon} \tag{5}$$

$$\frac{\partial h}{\partial y} = \frac{(h_{y+\epsilon} - h_{y-\epsilon})}{2\epsilon}$$

$$\frac{\partial h}{\partial z} = \frac{(h_{z+\epsilon} - h_{z-\epsilon})}{2\epsilon}$$

$$\vec{\nabla} h = \frac{\partial h}{\partial x}\hat{i} + \frac{\partial h}{\partial y}\hat{j} + \frac{\partial h}{\partial z}\hat{k} \tag{6}$$

where (x, y, z) represents the coordinate of the collision point and ϵ is a small parameter. To compute the gradient at the IHIP, we estimate the height values that are ϵ distance away from the IHIP along the coordinate axes. Texture heights can be estimated at these points using the two-stage mapping technique (Basdogan et al., 1997). For example, $h_{x+\epsilon}$ represents the estimated height value at the point $(x + \epsilon, y, z)$. Once the local gradient is known, it can be used to perturb the surface normal at the collision point for simulating image-based textures. Note that all the texture values indicated here are filtered values because of the need for C^0 and C^1 continuity.

(b) Procedural haptic texturing: The goal of procedural haptic texturing is to generate synthetic textures using mathematical functions. Generally speaking, it is much more straightforward to obtain the height value and the gradient of height in this approach. The function usually takes the coordinates (x, y, z) as the input and returns the height value and its gradient as the outputs. For example, we have implemented the well-known noise texture (Perlin, 1985; Ebert et al., 1994) to generate stochastic haptic textures. Fractals are also suggested for modeling natural textures since many natural objects seem to exhibit self-similarity (Mandelbrot, 1982). We have used the fractal concept in combination with the texturing functions mentioned above (e.g., Fourier series, noise) with different frequency and amplitude scales (Ebert et al., 1994) to generate more-sophisticated surface details. Other texturing techniques suggested in computer graphics can also be ported to generate haptic textures. For example, we have implemented haptic versions of reaction-diffusion textures (Turk, 1991; Witkin & Kass, 1991), the spot noise (Wijk, 1991), and cellular texture (Worley, 1996; Fleischer et al., 1995).

5 Results

We have successfully applied the rendering techniques proposed in this paper to render various objects both on a Windows NT platform and a Unix platform. In order to demonstrate the efficiency of our haptic interaction technique, we rendered 3-D models with varying complexities and properties. The structure of our

Table I. *Haptic Rendering Rates for Various 3-D Rigid and Smooth Objects*

	Object 1	Object 2	Object 3	Object 4
Number of vertices	8	239	902	32402
Number of lines	18	695	2750	97200
Number of polygons	12	456	1800	64800
Haptic servo rate (kHz)	~12 to 13	~11 to 12	~11 to 12	~9 to 10

The results were obtained with a Dual Pentium II, 300 MHz PC running two threads and equipped with an advanced graphics card (AccelECLIPSE from AccelGraphics). Servo rate results are based on rendering 3-D objects for at least 3 min. Rendering test is repeated at least three times for each object.

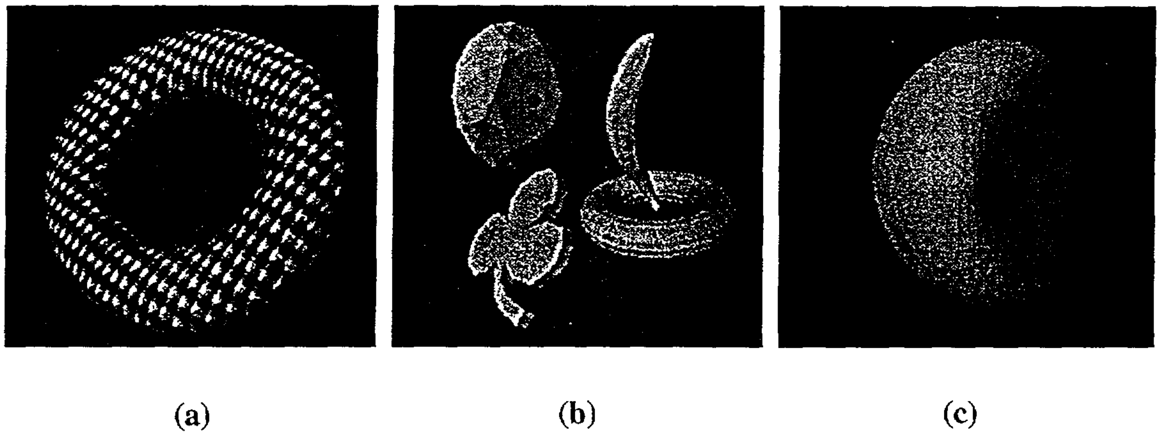

Figure 8. *Examples of haptically rendered 3-D objects that can be touched and felt through a haptic interface: (a) textured object, (b) objects with dynamics, (c) deformable object. The small ball in the figures represents the cursor (called the ideal haptic interface point, or IHIP)*

program includes two separate loops—one for updating graphics and the other for displaying force. The graphics update rate is maintained around 30 Hz, although it decreases slightly during the rendering of texture-mapped objects or deformable surfaces. We provide haptic rendering examples for four types of objects:

(a) *Rigid smooth objects:* The number of polygons of each polyhedron we tested ranges from hundreds to thousands. (See Table 1.) It can be seen that the haptic servo rate is approximately constant even if the number of polygons is increased by approximately 5,000 times.

(b) *Texture-mapped objects:* Figure 8(a) shows the simulation of a doughnut with texture mapping. In this simulation, an image-based haptic-texturing technique is used to map 2-D periodic waves onto the surface of the object. The users were able to feel the texture of the object when the surface of the object was explored with a haptic device.

(c) *Dynamic objects:* Multiple objects whose dynamics are governed by the equations of the motion are simulated. (See Figure 8b.) Dynamical equations are solved using the Euler integration technique, and the state of each object is updated at every iteration. The user can manipulate the objects to change their position or orientation dynamically via the end effector of the haptic device.

(d) *Deformable objects:* The vertices of the 3-D object that are in the close vicinity of the IHIP are moved, along the vector from IHIP to HIP, using a simple polynomial function to simulate deformable objects. Free-form deformation (FFD) techniques have also been implemented to change the topology of surfaces (Basdogan et al., 1998) (see Figure 8c).

6 Conclusions

In this paper, we have proposed a new point-based haptic-interaction technique that enables the user to touch and feel 3-D polyhedral objects and their surface details. The technique contains the "neighborhood watch" algorithm, which is capable of handling both convex and concave objects. After the first contact, the average rendering time is almost constant for complex objects independent of the number of polygons used to represent the object. We have also proposed methods to simulate surface properties of objects in virtual environments. In comparison to the existing haptic-texturing techniques, our texturing technique enables the mapping of textures onto 3-D surfaces. With the suggested models, friction and textures can be easily combined to generate realistic simulations. Moreover, the suggested haptic interaction technique can be easily extended to simulate the dynamics of rigid and deformable objects.

References

Adachi, Y., Kumano, T., & Ogino, K. (1995). Intermediate representation for stiff virtual objects. *Proc. IEEE Virtual Reality Annual Intl. Symposium '95,* 203–210.

Baraff, D. (1994). Fast contact force computation for nonpenetrating rigid bodies. *ACM (Proceedings of SIGGRAPH), 28,* 23–34.

Basdogan, C., Ho, C., & Srinivasan, M. A. (1997). A ray-based haptic rendering technique for displaying shape and texture of 3D objects in virtual environments. *ASME Winter Annual Meeting, 61,* 77–84.

Basdogan, C., Ho, C., Srinivasan, M. A., Small, S., & Dawson, S. (1998). Force interactions in laparoscopic simulations: Haptic rendering of soft tissues. *Proceedings of the Medicine Meets Virtual Reality VI Conference,* 385–391.

Bier, E. A., & Sloan, K. R. (1986). Two-part texture mapping. *IEEE Computer Graphics and Applications,* 40–53.

Blinn, J. F. (1978). Simulation of wrinkled surfaces. *ACM (Proceedings of SIGGRAPH), 12*(3), 286–292.

Burdea, G. (1996). *Force and Touch Feedback for Virtual Reality.* New York: John Wiley and Sons, Inc.

Chen, J., DiMattia, C., Falvo, M., Thiansathaporn, P., Superfine, R., & Taylor, R. M. (1997). Sticking to the point: a friction and adhesion model for simulated surfaces. *Proceedings of the Sixth Annual Symposium on Haptic Interfaces and Virtual Environment and Teleoperator Systems,* 167–171.

Cohen, J., Lin, M., Manocha, D., & Ponamgi, K. (1995). I-COLLIDE: An interactive and exact collision detection system for large-scaled environments. *Proceedings of ACM Interactive 3D Graphics Conference,* 189–196.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., & Worley, S. (1994). *Texturing and Modeling.* Cambridge, MA: AP Professional.

Fleischer, K. W., Laidlaw, D. H., Currin, B. L., & Barr, A. H. (1995). Cellular texture generation. *ACM (Proceedings of SIGGRAPH),* 239–248.

Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1995). *Computer Graphics: Principles and Practice.* New York: Addison-Wesley.

Fritz, J., & Barner, K. (1996). Haptic scientific visualization. In J. K. Salisbury & M. A. Srinivasan (Eds.), *Proceedings of the First PHANToM Users Group Workshop,* MIT-AI TR-1596 and RLE TR-612.

Gottschalk, S., Lin, M., & Manocha, D. (1996). OBB-Tree: A hierarchical structure for rapid interference detection. *ACM (Proceedings of SIGGRAPH).*

Green, D. F., & Salisbury, J. K. (1997). Texture sensing and simulation using the PHANToM: Towards remote sensing of soil properties. *Proceedings of the Second PHANToM Users Group Workshop,* 19–22.

Ho, C., Basdogan, C., & Srinivasan, M. A. (1997). Haptic rendering: Point- and ray-based interactions. *Proceedings of the Second PHANToM Users Group Workshop.*

Hubbard, P. (1995). Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics, 1*(3), 219–230.

Lin, M. (1993). *Efficient collision detection for animation and robotics.* Unpublished doctoral dissertation, University of California, Berkeley.

Mandelbrot, B. (1982). *The Fractal Geometry of Nature.* W. H. Freeman.

Mark, W., Randolph, S., Finch, M., Van Verth, J., & Taylor, R. M. (1996). Adding force feedback to graphics systems: Issues and solutions. *Computer Graphics: Proceedings of SIGGRAPH'96,* 447–452.

Massie, T. H. (1993). *Initial haptic explorations with the Phantom: Virtual touch through point interaction.* Unpublished master's thesis: Massachusetts Institute of Technology.

Massie, T. H., & Salisbury, J. K. (1994). The PHANToM haptic interface: A device for probing virtual objects. *Proceedings of the ASME Dynamic Systems and Control Division, 55*(1), 295–301.

Max, N. L., & Becker, B. G. (1994). Bump shading for volume textures. *IEEE Computer Graphics and App., 4,* 18–20.

Minsky, M. D. R. (1995). *Computational haptics: The sandpaper system for synthesizing texture for a force-feedback display.* Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Minsky, M., Ming, O., Steele, F., Brook, F. P., & Behensky, M. (1990). Feeling and seeing: Issues in force display. *Proceedings of the symposium on 3D Real-Time Interactive Graphics, 24,* 235–243.

Mirtich, B. (1996). Impulse-based dynamic simulation of rigid body systems. Unpublished doctoral dissertation, University of California, Berkeley.

Mirtich, B., Canny, J. (1995). Impulse-based simulation of rigid bodies. *Proceedings of Symposium on Interactive 3D Graphics.*

Moore, M., & Wilhelms, J. (1988). Collision detection and response for computer animation. *ACM (Proceedings of SIGGRAPH), 22*(4), 289–298.

Morgenbesser, H. B., & Srinivasan, M. A. (1996). Force shading for haptic shape perception. *Proceedings of the ASME Dynamic Systems and Control Division, 58,* 407–412.

Perlin, K. (1985). An image synthesizer. *ACM SIGGRAPH, 19*(3), 287–296.

Ruspini, D. C., Kolarov, K., & Khatib, O. (1996). Robust haptic display of graphical environments. In J. K. Salisbury & M. A. Srinivasan (Eds.), *Proceedings of the First PHANToM Users Group Workshop.* MIT-AI TR-1596 and RLE TR-612.

———. (1997). The haptic display of complex graphical environments. *ACM (Proceedings of SIGGRAPH),* 345–352.

Salcudean, S. E., & Vlaar, T. D. (1994). On the emulation of stiff walls and static friction with a magnetically levitated input/output device. *ASME* DSC, *55*(1), 303–309.

Salisbury, J. K., Brock, D., Massie, T., Swarup, N., & Zilles, C. (1995). Haptic rendering: Programming touch interaction with virtual objects. *Proceedings of the ACM Symposium on Interactive 3D Graphics.*

Salisbury, J. K., & Srinivasan, M. A. (1997). Phantom-based haptic interaction with virtual objects. *IEEE Computer Graphics and Applications, 17*(5).

Siira, J., & Pai, D. K. (1996). Haptic texturing—A stochastic approach. *Proceedings of the IEEE International Conference on Robotics and Automation,* 557–562.

Smith, A., Kitamura, Y., Takemura, H., & Kishino, F. (1995). A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. *IEEE Virtual Reality Annual International Symposium,* 136–145.

Srinivasan, M. A. (1995). Haptic Interfaces. In N. Durlach & A. S. Mavor (Eds.), *Virtual Reality: Scientific and Technical Challenges.* National Academy Press. (pp. 161–187).

Srinivasan, M. A., & Basdogan, C. (1997). Haptics in virtual environments: Taxonomy, research status, and challenges. *Computers and Graphics, 21*(4), 393–404.

Turk, G. (1991). Generating textures on arbitrary surfaces using reaction-diffusion. *ACM (Proceedings of SIGGRAPH), 25*(4), 289–298.

Watt, A., & Watt, M. (1992). *Advanced Animation and Rendering Techniques.* New York: Addison-Wesley.

Wijk, J. J. V. (1991). Spot noise. *ACM (Proceedings of SIGGRAPH), 25*(4), 309–318.

Witkin, A., & Kass, M. (1991). Reaction-diffusion textures. *ACM (Proceedings of SIGGRAPH), 25*(4), 299–308.

Worley, S. (1996). A cellular texture basis function. *ACM (Proceedings of SIGGRAPH),* 291–294.

Zilles, C. B., & Salisbury, J. K. (1995). A constraint-based god-object method for haptic display. *IEEE International Conference on Intelligent Robots and System, Human Robot Interaction, and Co-operative Robots,* IROS, *3,* 146–151.