Chih-Hao Ho
Cagatay Basdogan
Mandayam A. Srinivasan
Laboratory for Human and Machine Haptics
Department of Mechanical Engineering and
Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139, USA
http://touchlab.mit.edu

Ray-Based Haptic Rendering: Force and Torque Interactions between a Line Probe and 3D Objects in Virtual Environments

Abstract

Virtual environments (VEs) that enable the user to touch, feel, and manipulate virtual objects through haptic interactions are expected to have applications in many areas such as medicine, CAD/CAM, entertainment, fine arts, and education. The current state of technology allows the human operator to interact with virtual objects through the probe (such as a thimble or a stylus) of a force-reflecting haptic interface. Most of the current haptic interaction algorithms model the probe as a single point and allow the user to feel the forces that arise from point interactions with virtual objects. In this paper, we propose a ray-based haptic-rendering algorithm that enables the user to touch and feel convex polyhedral objects with a line segment model of the probe. The ray-based haptic-rendering algorithm computes both forces and torques due to collisions of the tip and/or side of the probe with multiple virtual objects, as required in simulating many tool-handling applications. Since the real-time simulation of haptic interactions between a 3D tool and objects is computationally quite expensive, the ray-based rendering can be considered as an intermediate step toward achieving this goal by simplifying the computational model of the tool. To compare the ray- and point-based haptic interaction techniques in the haptic perception of 3D objects, we conducted perceptual experiments in which the participants were asked to identify the shape of four different 3D primitives (sphere, cone, cylinder, and cube) that were displayed in random order using both point- and ray-based techniques. The results of the study show that on average, 3D objects are recognized faster with ray-based rendering than with point-based rendering.

KEY WORDS—virtual reality, haptic interface, rendering algorithm, force feedback, human-computer interaction

The International Journal of Robotics Research
Vol. 19, No. 7, July 2000, pp. 668-683,
©2000 Sage Publications, Inc.

1. Introduction

Enabling haptic interaction with 3D objects in virtual environments has been an exciting and challenging research topic for scientists and engineers during the past few years (Srinivasan and Basdogan 1997). With the technology available today, the user cannot get the complete sensation of directly touching the virtual objects. Instead, the user touches and manipulates virtual objects through an end-effector of a haptic interface device. This end-effector, referred to as a probe in the sections below, could be a thimble in which the fingertip could be inserted, a stylus or a mechanical tool that could be held in the hand, or an actuated glove or exoskeleton that the user could wear. Typical functions of the haptic device are to acquire the position of the probe and reflect forces arising from interactions with the virtual environment back to the user. For a given position information, collision detection software checks whether the probe is inside the virtual objects or not. If so, based on the shape and other physical properties of objects that are being simulated, it sends appropriate force commands to the motors of the device. With proper device design, suitable algorithms, and fast computations, it is possible to convey to the user a natural feel of the static and dynamic properties of objects (Salisbury and Srinivasan 1997).

In general, the physical probe can have different physical shapes and designs, and the same is true of its representation in probe-object collision detection and force response computations. However, for the feel of objects to appear natural to the user, it has been generally observed that the force update rates need to be of the order of kHz. Simplifications are therefore necessary to reduce the computational burden associated with collision detection and response. Most of the haptic rendering techniques developed so far are point based in which the probe is simply represented as a point. Zilles and Salisbury (1995);

Ruspini, Kolarov, and Khatib (1997); Adachi, Kumano, and Ogino (1995); Mark et al. (1996); Gregory et al. (1999); and Ho, Basdogan, and Srinivasan (1999) have proposed point-based interaction algorithms to render 3D polyhedral objects. Avila and Sobierajski (1996) have proposed techniques to render 3D volumetric objects. Salisbury and Tarr (1997) have proposed a method to render implicit surfaces. Thompson, Johnson, and Cohen (1997) have proposed a method for rendering NURBS surfaces. In all of the point-based rendering techniques, the user is able to explore the shape, surface details, and material properties of the virtual objects through the tip of the probe (Chen et al. 1997; Fritz and Barner 1996; Green and Salisbury 1997; Minsky et al. 1990; Salcudean and Vlaar 1994; Siira and Pai 1996; Basdogan, Ho, and Srinivasan 1997; Ho, Basdogan, and Srinivasan 1999). Since the probe is represented by a point, the net force at that point is the only haptic feedback variable that can be displayed to the user. For exploring the shape and surface properties of objects in VEs, these methods are probably sufficient and could provide the users with similar force feedback as what they would feel when exploring the objects in real environments with the tip of a stick.

Point-based methods, however, are not capable of simulating more general tool-object interactions that involve single or multiple objects in contact at arbitrary locations of the tool. In such a context, both forces and torques displayed to the user need to be independently computed. Furthermore, depending on the nature of the object in contact, the forces and torques can be dependent on the orientation of the probe. To handle such cases in the simplest possible manner, we propose ray-based rendering where the probe is represented by a finite line segment. In this algorithm, contact points, depth of penetration, and the distances from the contact points to both ends of the probe are computed first, followed by the computation of both the forces and torques that will be displayed to the user. Thus, ray-based rendering provides a basis for displaying not only the forces at the probe tip but also the forces and torques that are appropriate for any number of contacts along the long axis of the probe.

Modeling haptic interactions between a probe and objects using ray-based technique has several advantages over the existing point-based techniques. First, side collisions between the simulated tool and the 3D objects can be detected. The user can rotate the haptic probe around the corner of the object in continuous contact and get a better sense of the object's shape. In point-based methods, one of the common unrealistic feelings is that the probe and the user's hand can slice through objects without any resistance as long as the probe tip is outside the object (see Fig. 2). Ray-based rendering eliminates such a situation. Second, the ray that represents the probe can be extended to detect the collision path with multiple layers of an object. This is especially useful in haptic rendering of compliant objects (e.g., soft tissue) or layered surfaces (e.g., earth's soil), where each layer has different material properties and the forces/torques to be applied on the user depend on the probe orientation. Third, it enables the user to touch and feel multiple objects at the same time. If the task involves the simulation of haptic interactions between a tool and an object, ray-based rendering provides a more natural way of interacting with objects. Fourth, the reachable haptic workspace can potentially be extended using this technique since we have the full control of forces and torques that are displayed to the user. This means that it may be possible to create an illusion of touching distant objects by virtually extending the length of the probe and appropriately changing the direction and magnitude of the reflected forces (similar to seeing distant objects with a flashlight, see Fig. 3b).

The advantage of ray-based rendering over point-based rendering is quite apparent in some applications. For example, in performing minimally invasive surgeries, the surgeon inserts thin long rigid tubes into the body of the patient through several ports. Small instruments attached to these tubes are used for manipulating the internal organs. During the surgery, the surgeon accesses the targeted area by pushing the organs and surrounding tissue aside using the instruments and feels both the interaction forces and torques. A point-based technique is inadequate to fully simulate such haptic interactions between surgical instruments and virtual organs. If the instrument is modeled as a single point, the side collisions of an instrument with organs will not be detected and the instrument will pass through any organ other than the one touching the tip. We have observed that ray-based rendering significantly improves the realism of the simulation of haptic interactions between laparoscopic instruments and organs (Basdogan et al. 1998). In addition, multilayered and damaged tissues whose reaction forces depend on the tool orientation can be simulated better using the ray-based technique if the ray is extended through the contacted surface and multiple collisions with the layers of the virtual object are detected to compute interaction forces.

Another example where the ray-based rendering is preferable would be the simulation of assembly line in car manufacturing. A scenario may involve a mechanic going under a virtual car and turning the nuts of an engine block. Some of these procedures are done through mechanical instruments attached to a long and rigid shaft that enables the mechanic to reach difficult areas of the engine. Typically, the view of the workspace is limited and the mechanic finds his way around using haptic cues only. Moreover, the path to the nuts is usually blocked by several other mechanical components, which makes the haptic task even more challenging. The simulation of this procedure in virtual environments will certainly involve correct computation of torques and detection of multiple collisions simultaneously since a long rigid shaft is used to reach the targeted areas.

Although single-point interactions are not sufficient for computing the correct forces and torques generated due to contact between 3D objects, a group of points can sufficiently approximate the contact conditions. For example, McNeely, Puterbaugh, and Troy (1999) presented a voxel-based approach for 6-DoF haptic rendering with applications in the design for assembly and design for maintenance. In their approach, static objects in the scene are divided into voxels and the probe is modeled as a set of surface points. Then multiple collisions are detected between the surface points of the probe and each voxel of the static object to reflect forces based on a tangent-plane force model. A tangent plane whose normal is along the direction of the collided surface point is constructed at the center of each collided voxel. Then, the net force and torque acting on the probing object is obtained as the summation of all force/torque contributions from such point-voxel intersections. Although this approach enables 6-DoF haptic interactions with static rigid objects, its extension to dynamical and deformable objects would significantly reduce the haptic update rate because of the computational load. Moreover, rendering of thin or small objects will have difficulties with this approach.

In this paper, we have further improved and extended the ray-based rendering concept proposed in our earlier studies (Basdogan, Ho, and Srinivasan 1997) to include the computation of torques and modeling of various contact conditions. Moreover, the haptic rendering rate was significantly improved using hierarchical trees and a local search technique. Since the probe is modeled as a line segment, the proposed ray-based rendering method requires a haptic device that is capable of sensing at least 5 DoF (3 positions and 2 orientations) and delivering 5 degrees of force feedback (force feedback along 3 axes and torque feedback about 2 axes). To display torques as well as forces to the user, we have connected two commercially available force-feedback devices (PHANTOM from SensAble Technologies, Woburn, MA) to each other through a rigid probe. The result is similar to that achieved by Iwata (1993). With this extension, the haptic feedback conveyed to the user can be any combination of force and torque (see Fig. 1).

In the next section, we first describe our setup for displaying forces as well as torques to the user. We then introduce the concept of haptic rendering with a line segment in Section 3. Our collision detection algorithm for detecting collisions between a line segment model of the probe and the 3D convex objects is described in Section 4. In Section 5, we discuss the collision response phase that mainly deals with the computation of interaction forces and torques. In Section 6, we present an experimental study that is aimed to investigate the difference between point- and ray-based rendering techniques in perceiving the shape of 3D objects. The results and conclusions of the study are summarized in Section 7.

2. Hardware Setup

To implement our algorithm, we have put together a setup that is capable of displaying torques as well as forces (see Fig. 3). The haptic devices designed by Millman and Colgate (1991), Iwata (1993), and Buttolo and Hannaford (1995) are examples of such devices. The haptic device that we have used in our simulations is a commercial product available in the market (PHANTOM from SensAble Technologies, Woburn, MA). Each of the Phantoms we used can sense six degrees of position information but reflect forces along three axes only (i.e., torques cannot be displayed). However, if two Phantoms are connected to each other through a rigid probe, then a 5-DoF force/torque display can be obtained (see Fig. 3). This configuration, which runs on a PC platform (a dual 300 MHz Pentium II processor), is the one we used to implement ray-based rendering and to conduct experiments described in this paper. The software code for visual and haptic rendering of 3D polyhedral objects made of triangles is written with Open Inventor Graphics Tool Kit (TGS Inc.) and C++ programming language. Multithreading techniques were used to synchronize the visual and haptic loops to run the simulations in an efficient manner (Ho, Basdogan, and Srinivasan 1999).

3. Ray-Based Haptic Interactions

In ray-based rendering, we model the generic probe of the haptic device as a line segment and then detect collisions with 3D objects in the scene to compute the interaction forces/torques. As mentioned in the previous paragraphs, the existing haptic rendering techniques only consider the point-based interactions that take place between the end point of the haptic device and 3D virtual objects. In the point-based case, the point probe and virtual object can only have point-vertex, point-edge, or point-polygon contacts (Ho, Basdogan, and Srinivasan 1999). However, the type of contacts between the line segment model of the haptic probe and virtual object can, in addition, be line segment-edge and line segment-polygon. There can also be multiple contacts composed of a combination of the above cases.

To compute the force to be reflected to the users, it is necessary to know which face of the object was penetrated. Hence, one has to consider the history of the probe's movements for proper collision response to the user in haptic rendering, which is one of the main differences from graphical rendering. Although this means the tracking of probe's tip position in the point-based interactions, the tracking of probe's orientation has to be considered as well in ray-based interactions. Therefore, the detection of collisions between a line-segment model of a haptic probe and arbitrary shaped 3D objects (i.e., convex and concave) is more complicated and computationally more expensive than that of point-based interactions. However, the advantages of the ray-based rendering over the point-based techniques that are mentioned in the introduction section were quite appealing to us. We have developed a rule-based algorithm that can successfully handle the collision conditions between a line segment model of

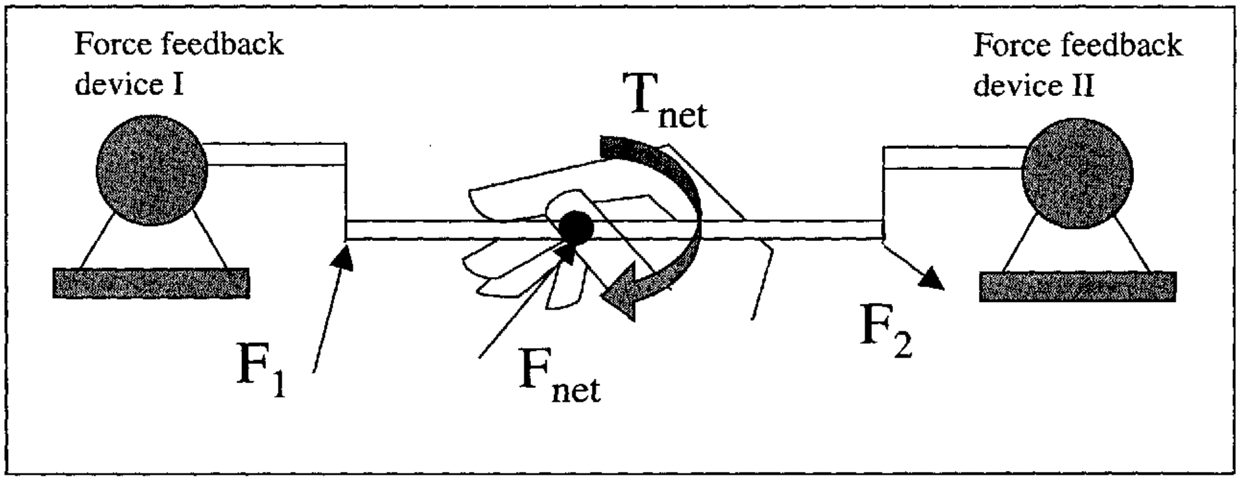

Fig. 1. Schematic of our haptic display: To display forces and torques to the user, we have connected two force feedback devices to each other through a rigid probe. This enables us to display forces in three axes and torques about two axes (the torsion about the long axis of the probe cannot be displayed with this design).

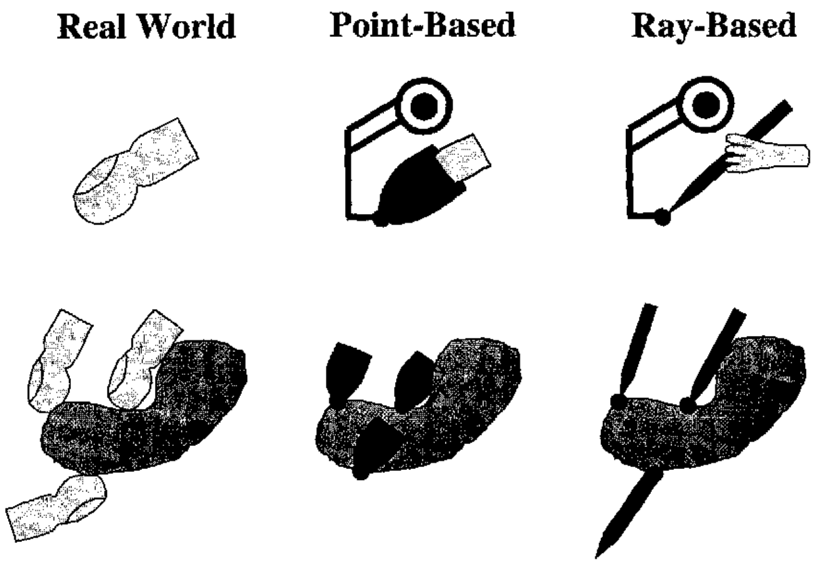

Fig. 2. The difference in point- and ray-based rendering techniques: In point-based rendering, the haptic probe is modeled as a single point leading to artifacts such as feeling the bottom surface of the object by passing through the object. In ray-based rendering, collisions between virtual objects and end-points as well as side of a line segment are detected.

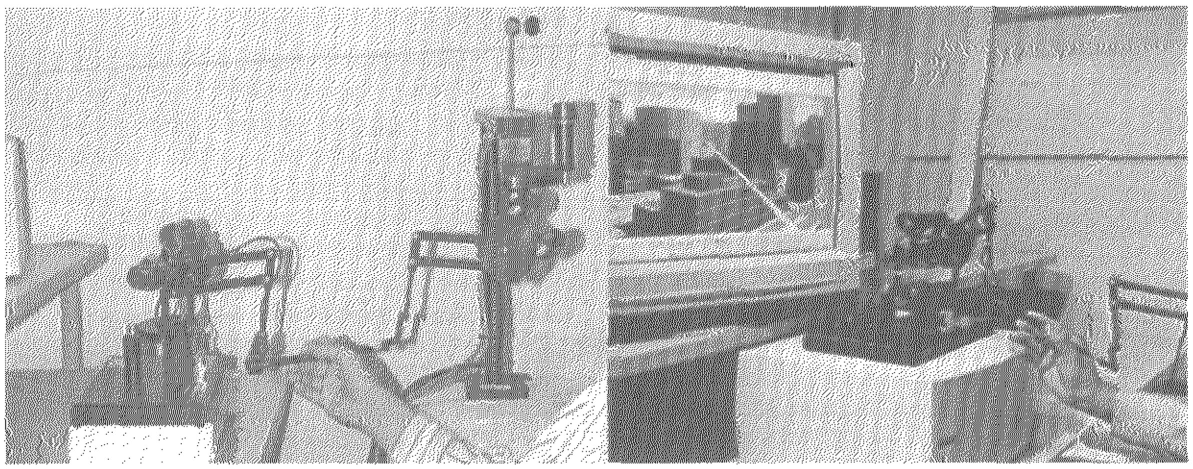

Fig. 3. (a) Our two-Phantom setup for displaying forces and torques to the user by employing the ray-based haptic rendering algorithm. (b) The concept of touching and exploring distant objects by using the ray-based rendering algorithm to extend the length of the haptic probe virtually.

a haptic probe and triangular convex objects. Although the algorithm can successfully render only convex objects at this stage, we do not consider this as a major limitation since the concave objects can be divided into several convex objects.

In general, three types of collisions occur when a convex object is explored with a line probe: (1) either end of the probe contacts with a polygon (we use triangles in our implementation) of the polyhedron and the intersection is a single point (point-polygon, see Fig. 4a); (2) the probe collides with an edge of the polyhedron and the intersection is a single point that is between the end points of the probe (line segment-edge, see Fig. 4b); (3) the probe is perfectly parallel to a polygon or a set of polygons of the object in contact, and the intersection is a portion of the probe (line segment-face, see Fig. 4c). Other types of contacts such as point-vertex, point-edge, and line segment-vertex are very unlikely to happen and will be covered by the three contact types mentioned above because the boundaries are included in the definition of an edge or a face.

For haptic rendering of objects in VEs, in addition to collision detection, a "collision response" phase has to be considered to calculate and reflect reaction forces and torques to the user. To better describe the steps of the collision response phase, we refer to three different definitions of the haptic probe in the text: (1) real probe, the physical piece that is held by the user; (2) virtual probe, the computational model of the probe (i.e., line segment model) that is defined by the tip and tail coordinates of the real probe; (3) ideal probe, the ideal location of the virtual probe that is constrained to stay on the surface when the virtual one penetrates the object (similar to the "God-Object" proposed by Zilles and Salisbury 1995). As we manipulate the physical probe of the haptic device in the real environment, the line segment model of the probe (i.e., virtual probe) that is defined by the end points of each Phantom is updated at each servo loop. The collisions of this line segment with the virtual objects are then detected. Although the line segment model of the virtual probe can be anywhere in 3D space, the movements of the ideal probe are restricted such that it cannot penetrate into objects. The location of the ideal probe relative to the virtual one is displayed in Figure 5 for each contact condition.

To properly distribute the forces to each force feedback device, we need to know the point of contact on the ideal stylus (see Fig. 5). The net moment is computed with respect to this point to distribute the forces. The location of this point on the ideal stylus changes with respect to the contact type. For example, in point-polygon collision (Fig. 5a), the contact point coincides with the tip point of the ideal probe. On the other hand, in line segment-face contact (Fig. 5c), the location of the equivalent contact point depends on the portion of the probe that is in contact with the surface. The details of the collision detection and the response phases are described in the following sections.

4. Collision Detection

At present, we have constrained the ray-based rendering method to handle only convex objects in order to simplify the computations. In fact, this simplification permits us to use our local search technique (called "Neighborhood Watch") that is described in our earlier studies (Ho, Basdogan, and Srinivasan 1999). Finding the constrained position and orientation of the ideal probe using a localized neighborhood search makes the rendering rate independent of the number of polygons. In addition, we construct two types of hierarchical databases for each object in the scene to detect collisions faster: (1) a bounding-box and (2) a neighborhood-connectivity hierarchical tree. (Similar techniques have been extensively used in computer graphics, such as by Lin 1993; Gottschalk, Lin, and Manocha 1996; Cohen et al. 1995.) The bounding box hierarchy is used for detecting the first collision between the virtual probe and virtual objects. The neighborhood connectivity

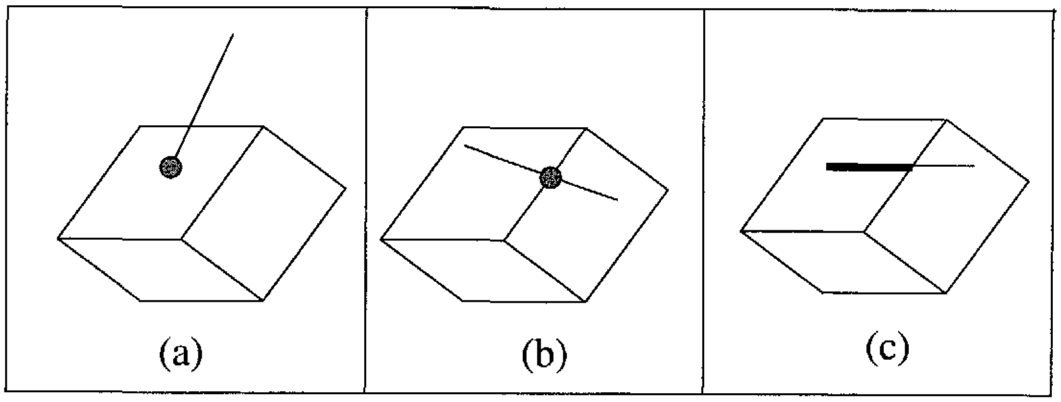

Fig. 4. Possible contact conditions for the line segment-convex object interactions: (a) the probe contacts a polygon of the object at a single point (point-polygon), (b) the probe intersects an edge of the convex object at a single point (line segment-edge), (c) the probe stays on the surface of a plane constructed from a single or multiple polygons, and the intersection between the probe and the plane is either a portion or all of the probe (line segment-face).

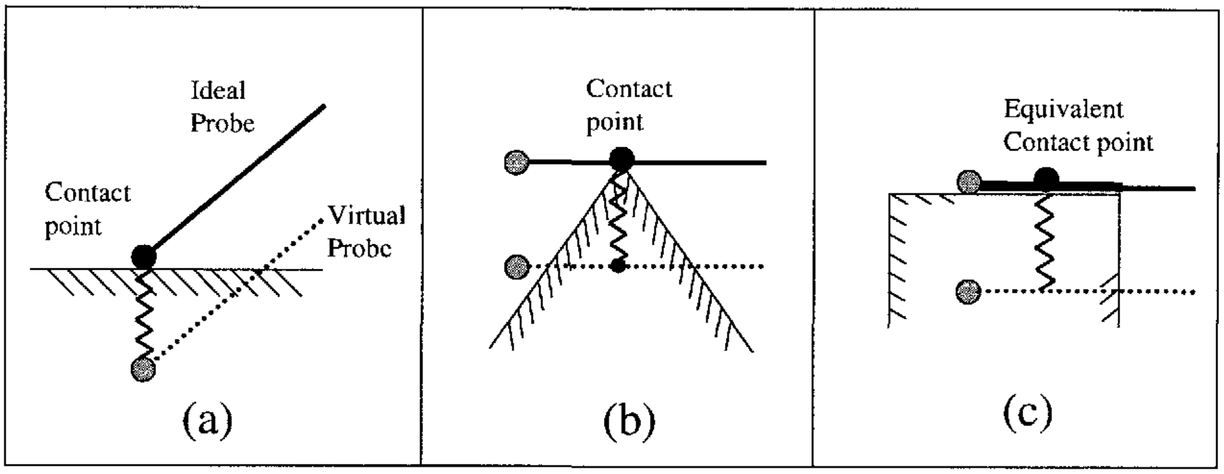

Fig. 5. Location of ideal probe relative to the virtual one for the contact conditions shown in Figure 4. Although the virtual probe can penetrate into objects, the ideal one is restricted to stay outside the objects. During the interactions, the collisions of the virtual probe with objects are detected and the collision information is used to calculate the location of the ideal probe. The stretch of the spring between the ideal and virtual probes in the figure illustrates a method for computing force interactions between the probe and the convex object for three different contact conditions.

is used to perform local searches for finding the subsequent contacts, which significantly reduces the computational time. Once the contact type is identified, we can compute the reaction forces using the "collision response" method described in Section 5.

In general, the collision detection phase between the virtual probe and a virtual object is composed of two cases: the probe did or did not have a collision with the object in the previous loop. Once this is known, the following steps are followed to detect the subsequent contacts:

a) If there was no contact with the object in the previous cycle, we first check (1) if the movement of end points of the line probe causes contact with any polygon of the object (see the appendix, Section 7, for details), or (2) if the movement of the line probe crosses any edge of the object (see the appendix, Section 8, for details). To speed up the collision detection calculations, we create two separate hierarchical trees (i.e., for the polygons and the edges of the object, respectively) for each polyhedron in the scene during the precomputation phase. At the top of the hierarchy is a bounding-box that covers the whole object. All of the geometric primitives inside the bounding-box are then separated into two groups based on their geometric centers (Gottschalk, Lin, and Manocha 1996). We then create a bounding-box for each group. These two new bounding-boxes are placed under the first bounding-box in the hierarchy. We repeat the same process to create two children for each parent at each hierarchical level of the tree until there is only one primitive left in each bounding-box. To detect a collision, we first check if the line probe is inside the top-most bounding-box of the hierarchical tree. If so, we then check the collisions with the bounding-boxes at the second level. This process is repeated until the lowest level is reached by progressing along a particular branch of the tree. Finally, we check if the line probe collides with the primitive itself (i.e., polygon or edge) that is inside the lowest-level bounding-box of this branch. If either of the end points of the virtual probe penetrates a polygon, we have a point-polygon collision. If the virtual probe crosses any edge, we encounter line segment-edge collision.

b) If there was a certain type of contact with the object in the previous cycle (i.e., continuation of the previous contact) such as point-polygon, line segment-edge, or line segment-face, we then study all the possible contact conditions for the upcoming loops. For example, if the contact type in the previous cycle was point-polygon, then the possible contact conditions for the current loop can be point-polygon, line segment-edge, or line segment-face. Each contact condition and the possible contact conditions that may follow it are discussed in detail below.

1. If there was a point-polygon collision in the previous loop: The first step is to update the vector that defines the probe. A line segment that connects the end points of the probe defines this vector ($\vec{V}$), and its direction is from the end that was in contact with the object in the previous loop to the other end point (see Fig. 6). We then calculate the dot product of $\vec{V}$ and the normal ($\vec{N}$) of the polygon that was contacted in the previous loop.

   i) If the dot product of the collided polygon normal ($\vec{N}$) and the vector ($\vec{V}$) is larger than zero (i.e., if the angle between these two vectors is less than 90 deg), we first project the collided end point of the probe to the plane of the previously collided polygon (see the appendix, Section 1) and then check if the projected point is still inside the same polygon in the current loop. If so, the type of contact condition in the current servo loop is a point-polygon collision again (Fig. 7a). If not, the probe could have a point-polygon collision with a neighboring polygon (Fig. 7b) or have a line segment-edge collision with a neighboring edge (Fig. 7c). If the collided end point is above the surface containing the polygon, then there is no collision in the current cycle.

   ii) If the dot product of ($\vec{N}$) and ($\vec{V}$) is smaller than zero, we encounter a line segment-face contact (Fig. 7d).

2. If there is a line segment-edge collision in the previous loop: First, we find the projection of the probe on a plane that contains the previously contacted edge and whose normal is perpendicular to the probe (see the appendix, Section 4, for details). Then, we check whether the

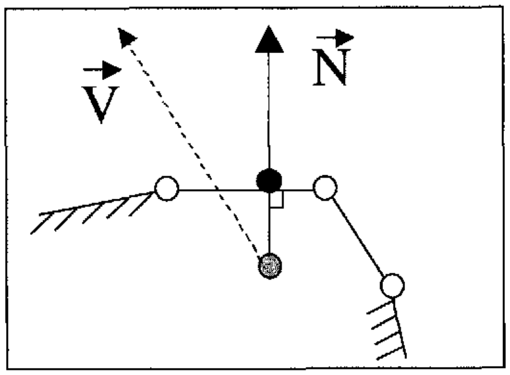

Fig. 6. A point-polygon collision: In this figure, one end of the probe is already inside the object whereas the other end point is outside the object.

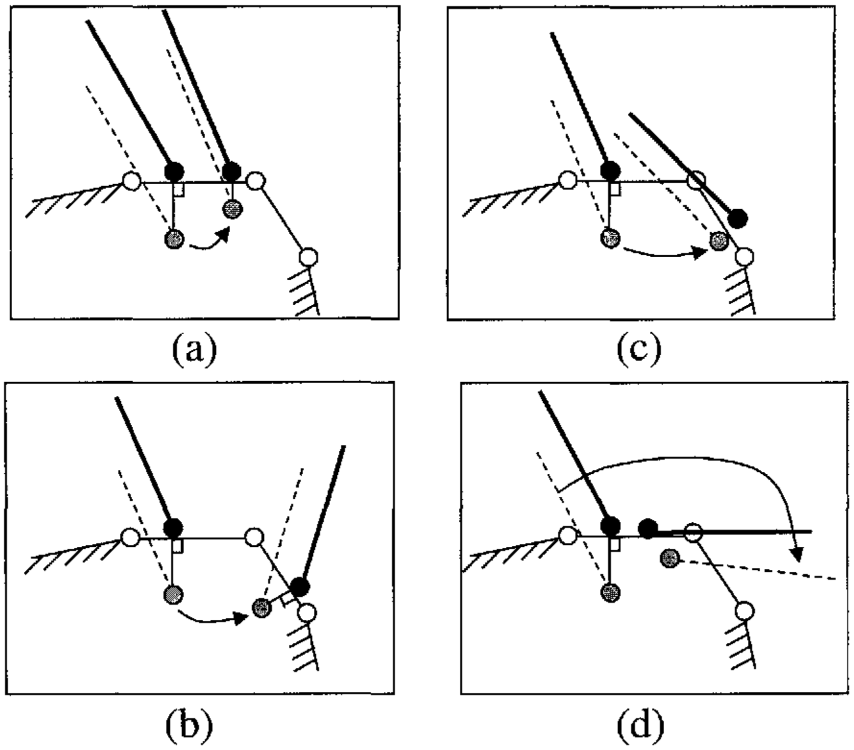

Fig. 7. Possible contact types following a point-polygon collision: (a) point-polygon, (b) point-neighboring polygon, (c) line segment-edge, and (d) line segment-face.

projected probe has an intersection with the previously collided edge. In addition to this check, we also define two angles (α and β) to describe the collision status of the probe with the edge (see Fig. 8a). Each edge primitive in the database has two neighboring polygons. The angle β is the angle between the first polygon (arbitrarily chosen) and the extension of the second polygon (see Fig. 8a). Similarly, the angle α is the angle between the first polygon and the probe.

i) If the value of α is larger than zero and smaller than the value of β and the projection of the probe (see the appendix, Section 4) has an intersection with the edge of the object, the probe should still be in contact with the same edge. If the probe is above the edge, then there is no contact.
ii) If the value of α is larger than zero and smaller than the value of β and the projection of the probe (see the appendix, Section 4) does not have an intersection with the edge, the probe should either have a line segment-edge collision (see Fig. 8b), a point-polygon collision (see Fig. 8c), or no collision at all in the upcoming loop.
iii) If the value of α is smaller than zero or larger than the value of β, we infer that the probe has a line segment-face contact with a neighboring face.

3. If there is a line segment-face collision in the previous loop: We have already discussed the cases in which the tip of the probe penetrates the object (point-polygon) and the probe collides with the edge of the object (line segment-edge). However, there is a situation, for example, in which a part of the probe may be on the surface of the object. Or, when we touch the surface of a convex object with the tip of a probe and then rotate the probe around the contacted polygon in continuous contact: first, a single-point contact occurs (point-polygon), then the probe becomes parallel to the contacted surface of the object. We call the phase described in these examples as line segment-face where the term *face* refers to the face that is constructed from the collided polygon (i.e., since the probe lies on the surface of the object, it could be in contact with multiple polygons). For the detection of line segment-face contact, we first define an angle (θ) that is between the probe and the face (see Fig. 9). We then check whether the

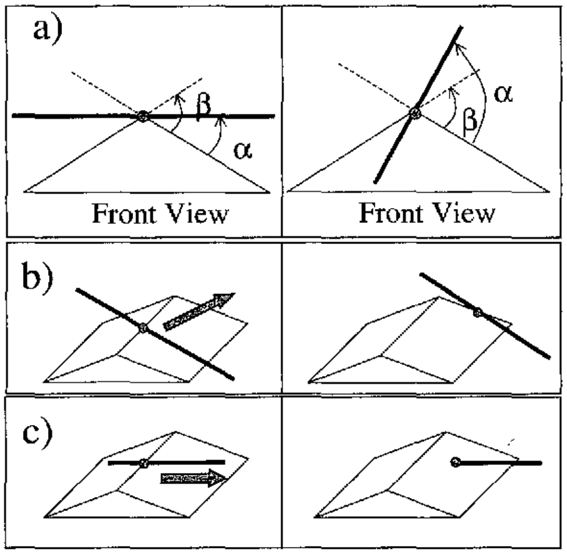

Fig. 8. Possible collision situations following a line segment-edge contact.

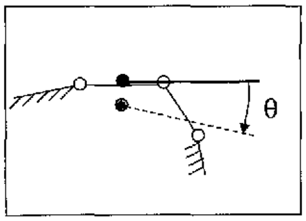

Fig. 9. Line segment-face collision.

projection of the probe on the plane of the face (see the appendix, Section 3) still has a collision with the face or not.

i) If the value of θ is smaller than a user-defined small angle epsilon (set at, say, 1 deg) and the projection of the probe (see appendix, Section 3) has a collision with the face, the probe should be in contact with the same face. If the probe is above the face, then there is no contact.
ii) If the value of θ is larger than epsilon and the projection of the line probe (see appendix, Section 3) has a collision with the face, the type of contacts can be a point-polygon (see Fig. 10a), line segment-edge (see Fig. 10b), or no collision at all.
iii) If the projection of the probe (see appendix, Section 3) does not have a collision with the face, we trace the path of the probe and find out which direction the probe moves out of face. Based on this direction, we determine whether the probe has a point-polygon collision, a line segment-edge collision, or no collision at all. To determine the type of contact, we use the direction of the movement in the following manner: (a) If the probe moves out of the face from an edge, we check if the end points of the probe are below the neighboring polygon of this edge. If so, then there is a point-polygon collision (see Fig. 11), otherwise there is no collision. (b) If the probe moves out of the face through a vertex, we check if the probe is below any of the neighboring edges of the vertex (see Fig. 11). If so, then there is a line segment-edge collision. To quickly determine the unqualified edges (i.e., a vertex can have multiple neighboring edges) in which the probe cannot possibly collide, we consider an imaginary plane. This plane contains the vertex in consideration, and its normal is determined using the closest vector from the vertex to the projected probe (i.e., the probe is projected to the plane

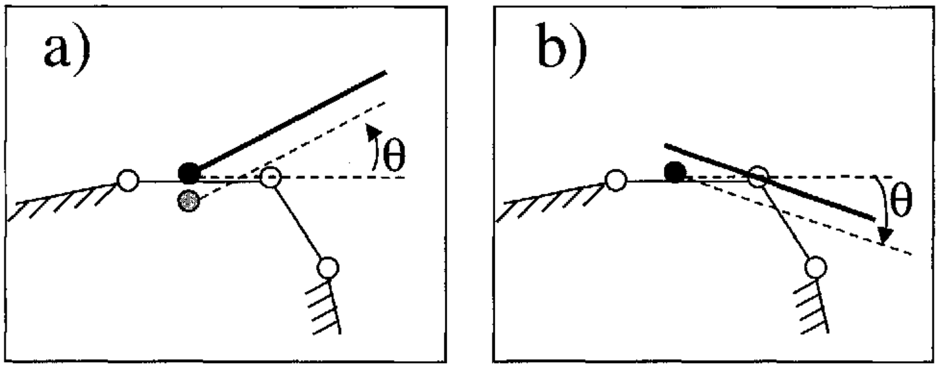

Fig. 10. Possible future contacts if θ becomes larger than epsilon and the projection of the line probe has a collision with the face in line segment-face contact: (a) point-polygon, (b) line segment-edge.

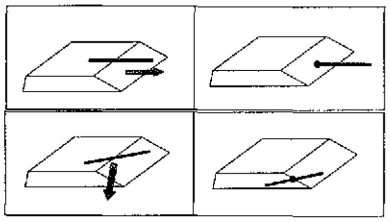

Fig. 11. Possible contact types if the projection of the probe (see appendix, Section 3) does not have a collision with the face when there is a line segment-face contact in the previous cycle: (a) point-polygon and (b) line segment-edge.

of face). If the neighboring edges of the vertex are behind this imaginary plane, then we do not consider them for a possible collision.

5. Collision Response

In haptic rendering, collision response involves the computation of the ideal probe location relative to its virtual counterpart and the reaction forces/torques that arise from interactions with 3D objects. Although the collision response phase has been studied in computer graphics (Baraff 1994; Mirtich and Canny 1995; Moore and Wilhelms 1988), the computations relevant to haptics are quite different (Ho, Basdogan, and Srinivasan 1999).

- The location of the ideal probe relative to the current location of the virtual probe: The probe that is held by the user is free to move anywhere in 3D space until its motion is constrained by force feedback or workspace boundary of the device. Since the virtual probe is allowed to penetrate into objects for the purpose of detecting collisions, we have to compute the location and orientation of the ideal probe for the calculation of forces/torques (see Fig. 5).
- Computation of forces and torques that will be displayed to the user: During haptic interactions with virtual objects, the computer sends force commands to the haptic device. This prevents the user from further penetrating into the objects. The forces and torques are computed based on the differences in the locations of the virtual and ideal probes.

The computation of ideal probe location and the interaction force depends on the type of contact. Each contact case is studied in detail below.

- In point-polygon collision, we first determine the surface point on the collided polygon that is nearest to the end point of the virtual probe. Then, the virtual probe is projected to this nearest surface point to define the new position of the ideal probe while keeping its orientation the same as the virtual probe. Following the projection, the nearest surface point and the end point of the ideal probe coincide. This nearest surface point is called the contact point in Figure 5a since it enables us to compute interaction forces and torques. We assume that there is a virtual spring between the contacted end points of the virtual and ideal probes (see Fig. 12a). The forces and torques are then computed based on the spring constant and the position difference between the virtual and ideal probes (see Ho, Basdogan, and Srinivasan 1999 for details).
- In line segment-edge collision, we first determine the plane that contains the collided edge and is parallel to the virtual probe. The virtual probe is projected to this plane (see appendix, Section 4) to define the new position of the ideal probe. We then compute the intersection point of the collided edge and the ideal probe, which is called the contact point in Figure 5b. To compute the net interaction force, we assume that there is a virtual spring between the virtual and ideal

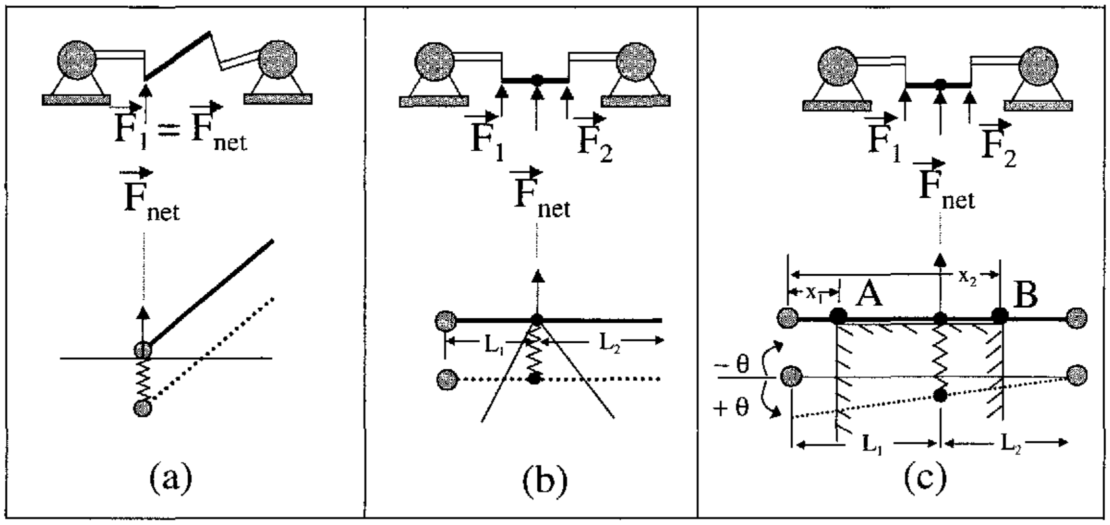

Fig. 12. The collision response for ray-based haptic rendering: The ideal location of the probe is determined based on the type of collision (i.e., point-polygon, line segment-edge, or line segment-face). In the figure, dashed and solid lines represent the virtual and ideal probes, respectively.

probes (see Fig. 12b). The net force is then distributed appropriately to the two Phantoms to display them to the user:

$$\vec{F}_1 = \frac{L_2}{L}\vec{F}_{\text{net}}, \quad \vec{F}_2 = \frac{L_1}{L}\vec{F}_{\text{net}}, \tag{1}$$

where L is the total length of the stylus ($L_1 + L_2$), $\vec{F}_1$ and $\vec{F}_2$ are the forces reflected from the two Phantom devices, and $\vec{F}_{\text{net}}$ is the net reaction force computed using, say, a spring-based force model (i.e., Hook's law).

- In line segment-face collision, a part of the ideal probe lies on the surface of the object while the virtual one penetrates into the object. If the user rotates the probe even slightly, the type of contact may quickly change to point-polygon or line segment-edge. This is undesirable since it can cause instability. For this reason, the orientation of the probe relative to the object surface (angle θ in the Fig. 9) plays an important role in computing the equivalent contact point where the net force is acting. We first determine the boundary points of the face that collides with the probe (note that the probe intersects the boundaries of the face at two points, which are marked as A and B in Fig. 12c). We then compute the distances from one end of the probe to these points (x_1 and x_2 in Fig. 12c). Now, we can compute the location of the collision point where the net force is acting as follows:

$$L_1 = \left(\frac{x_1 + x_2}{2}\right) + \left(\frac{\theta}{\theta_{\text{epsilon}}}\right)\left(\frac{x_1 - x_2}{2}\right), \tag{2}$$

where, $L_2 = L - L_1$ and the angle θ is defined as the current orientation of the probe relative to the collided and it varies between $-\theta_{\text{epsilon}}$ and θ_{epsilon}. In our simulations, θ_{epsilon} was chosen as one degree. For example, observe the collision response phases in Figure 12c: If θ is equal to θ_{epsilon}, then L_1 becomes equal to x_1 and the equivalent contact point moves to point A and the contact phase switches to line segment-edge. Similarly, if θ is equal to $-\theta_{\text{epsilon}}$, then L_1 becomes equal to x_2. The equivalent contact point moves to point B and the contact phase switches to line segment-edge. For θ between $-\theta_{\text{epsilon}}$ and θ_{epsilon}, L_1 will have a value between x_1 and x_2.

Following the computation of L_1 and L_2, the force that will be reflected from each Phantom can be easily determined using eq. (1).

In the situations above where contact with a single object is considered, the user will feel both forces and torques when grasping the probe at any point other than the contact point. Similarly, the user will in general feel torques as well as forces when there are multiple objects to interact in the scene and the movement of the probe is constrained. For example, if there are two convex objects and the user attempts to touch both of them at the same time as it is schematically described in Figure 13 and illustrated in an example in Figure 14, a certain amount of torque and force will be felt. The net force ($\vec{F}_{\text{net}}$) that will be reflected to the user's hand under this situation will be the vector summation of the forces $\vec{F}^a$ and $\vec{F}^b$ (see Fig. 13).

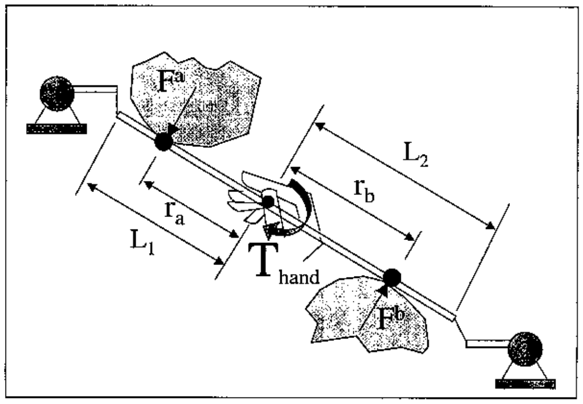

Fig. 13. Computation of interaction forces/torques when the line probe interacts with two virtual objects: The forces are distributed to the Phantoms based on the net force and moment principles.

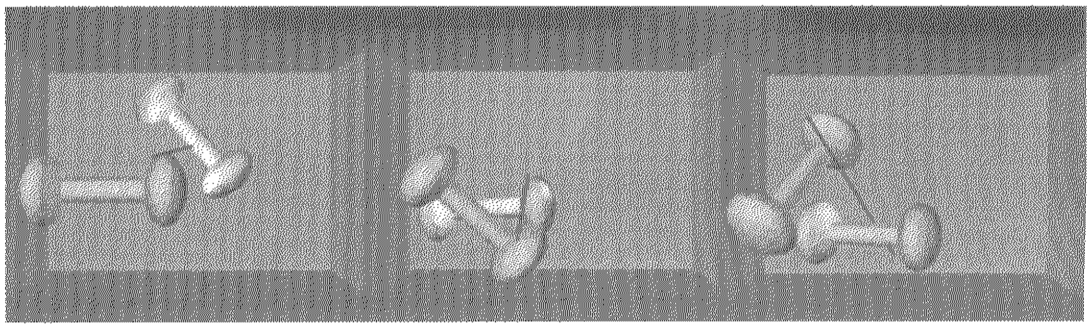

Fig. 14. The floating dumbbells: The user can touch, explore, and manipulate two dumbbells at the same time with a single probe using the ray-based rendering technique.

The net torque acting on the user's hand is computed as

$$T_{\text{hand}} = \vec{F}^a \times \vec{r}^a + \vec{F}^b \times \vec{r}^b. \quad (3)$$

The component of the force F^a that will be reflected from Phantom 1 and Phantom 2 can be computed using eq. (1) as

$$\vec{F}_1^a = \frac{L_2 + r^a}{L}\vec{F}^a, \quad \vec{F}_2^a = \frac{L_1 - r^a}{L}\vec{F}^a. \quad (4)$$

Similarly, the force $\vec{F}^b$ is distributed between the two Phantom devices as

$$\vec{F}_1^b = \frac{L_2 - r^b}{L}\vec{F}^b, \quad \vec{F}_2^b = \frac{L_1 + r^b}{L}\vec{F}^b. \quad (5)$$

6. An Experiment on the Haptic Perception of 3D Objects: Ray-Based versus Point-Based Interactions

To compare the effectiveness of the proposed haptic-rendering algorithm over the existing point-based rendering techniques in perceiving the shape of 3D objects, we designed and conducted a perceptual experiment. A total of 7 participants participated in the experiment. Participants were asked to identify the shape of 3D objects as quickly as possible using haptic cues only (i.e., no visual feedback was provided to the participants). Four primitive objects (i.e., sphere, cone, cylinder, and cube), rendered either with the point-based or the ray-based technique, were randomly displayed to the participants, one at a time and in various orientations. Each participant repeated the experiment for 304 times (i.e., each object was displayed 76 times) under each of the two different conditions for haptic interactions (i.e., point-based versus ray-based). We measured the total time each participant took to identify an object and then compared the results for point-based versus ray-based rendering conditions. The results indicate that haptic perception of 3D objects with the ray-based technique is faster than the point-based for the cone and cube (see Fig. 15). This is possibly because (a) the sphere and cylinder can be efficiently identified solely by exploring them with the probe tip and (b) the cone and cube can be identified faster when side collisions with the probe are rendered.

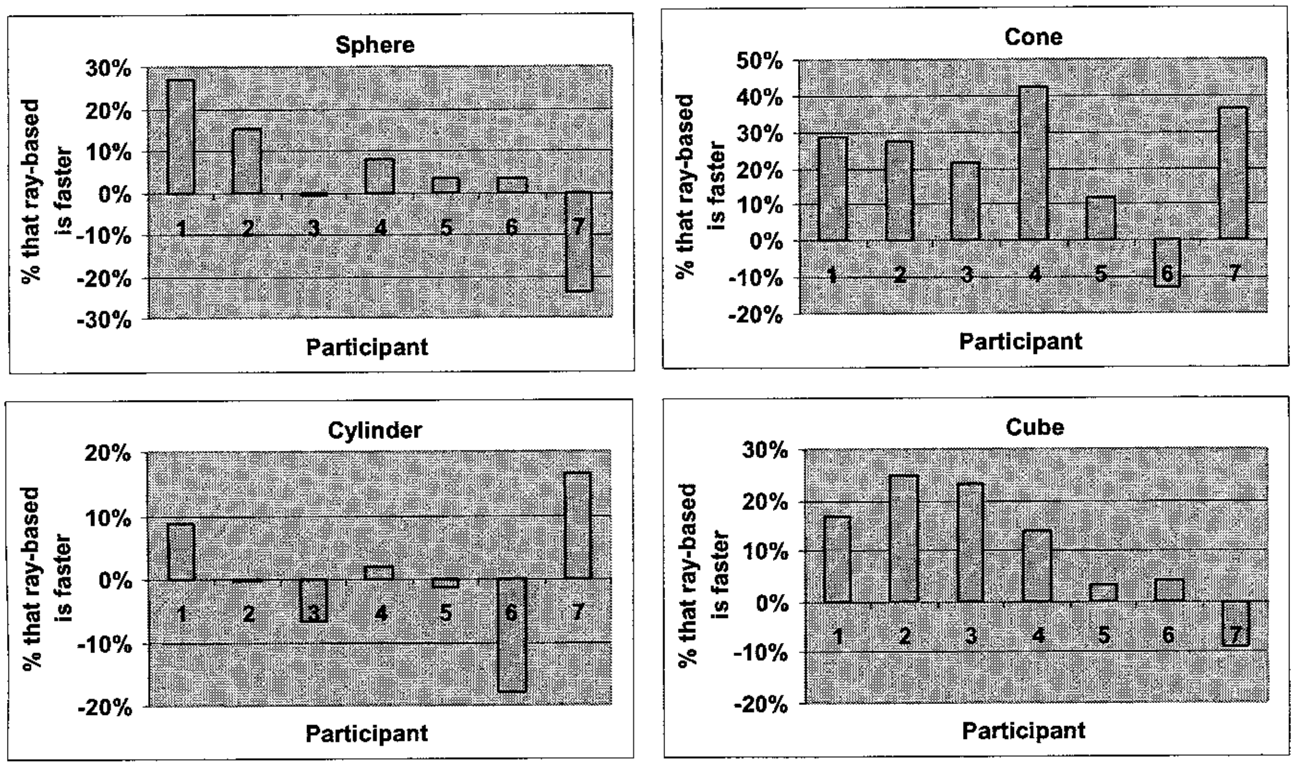

Fig. 15. Comparison of the point- and ray-based rendering techniques: The use of the ray-based technique leads to a faster perception of cone and cube (on the average, 22% and 11% faster than point-based, respectively). However, there were no significant differences for the sphere and cylinder.

7. Discussion and Conclusions

In this paper, we have proposed a ray-based rendering technique for simulating the haptic interactions between the end effector (i.e., probe) of a haptic interface and 3D virtual objects. In contrast to the point-based approaches that model the probe as a single point, the ray-based rendering technique models the probe as a line segment. Although the point-based approach simplifies the computations, it only enables the user to feel net forces due to contact with the probe tip. The ray-based haptic-rendering algorithm computes both forces and torques due to collisions of the tip and/or side of the probe with multiple virtual objects, as required in simulating many tool-handling applications. Moreover, as our experimental study demonstrates, the haptic perception of some 3D objects using the ray-based rendering technique is better than the existing point-based techniques when there are no visual cues available. Although the ray-based rendering algorithm described in this paper is for convex polyhedral objects only, it is capable of handling concave objects as well if they are represented as a combination of multiple convex objects. We have, for example, successfully rendered concave 3D objects using the ray-based rendering technique (see Figs. 14, 16, and 17).

The ray-based rendering algorithm can be superior to the point-based techniques in many applications. If we need to explore an interior surface of an object where visual cues are limited or if the interaction torques are important in executing a task as in assembly planning and design, ray-based rendering would be advantageous. Moreover, the ray-based rendering can be considered as an intermediate stage in progress toward the full 6-DoF haptic rendering. Since modeling of the haptic interactions between arbitrary-shaped 3D objects is computationally too expensive (especially in interacting with dynamical or deformable objects with a 3D haptic probe), an intermediate step for rendering both forces and torques will be helpful. For example, the computational model of a 3D mechanical tool can be easily constructed from a few line segments to achieve faster haptic rendering rates (see Fig. 17a). We have achieved real-time update rates for rendering dynamical and deformable objects using ray-based rendering, which would not be possible if full 3D object-object interactions were considered (see Fig. 17b).

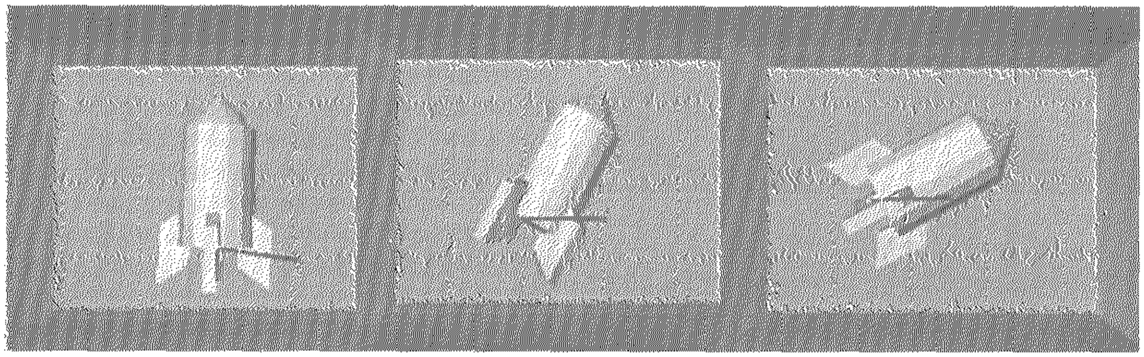

Fig. 16. The flying rocket: The ray-based rendering technique has been used to manipulate a rocket with concave parts that have been subdivided into multiple convex parts.

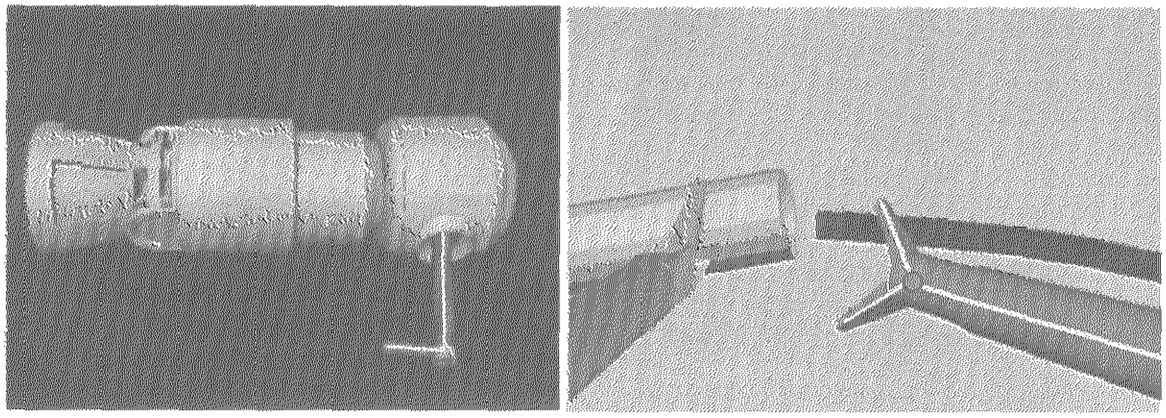

Fig. 17. (a) Haptic interactions between a mechanical tool and an engine block were simulated using the ray-based rendering technique. The engine block is made of convex objects, and the mechanical tool is modeled as two line segments shown in white. (b) Haptic interactions between surgical instruments and flexible objects were simulated using the ray-based technique. The computational model of the surgical instrument seen in the figure consists of the three white line segments.

Appendix

1. Projection of a Point to a Plane

Given a point p, a plane with a unit normal $\vec{N}$ and another point p_0 on the plane, the projected point p' of point p on the plane is calculated as

$$p' = p + ((p_o - p).\vec{N})\vec{N}. \tag{6}$$

2. Distance between a Point and a Plane

Given a point p, a plane with a unit normal $\vec{N}$ and another point p_0 on the plane, the distance between the point p and the plane is calculated as $d = \|(P_0 - P) \cdot \vec{N}\|$.

3. Projection of a Line Segment to a Plane

Given a line segment (P_a, P_b), a plane with a unit normal $\vec{N}$ and another point p_0 on the plane, we want to calculate the projected line segment (P'_a, P'_b) of the line segment (P_a, P_b) on the plane. Use the method mentioned in Section 1 of the appendix to project the point P_a to the plane and obtain the projected point P'_a. In the same way, we can obtain the projected point P'_b of the point P_b. Then, the line segment (P'_a, P'_b) is the projection of the line segment (P_a, P_b) onto the plane.

4. Projection of a Line Segment to a Plane Parallel to the Line Segment and Containing Another Line Segment

Two line segments (P_a, P_b) and (P_c, P_d) are defined in the 3D space. First, we find a vector $(\vec{N})$ that is perpendicular to the two line segments (P_a, P_b) and (P_c, P_d) (i.e., cross product of two vectors (P_a, P_b) and (P_c, P_d)) and normalize it. The plane parallel to the line segment (P_a, P_b) and containing the line segment (P_c, P_d) will have the unit normal $\vec{N}$ and the point P_c. Use the method mentioned in Section 3 of the appendix to project the line segment (P_a, P_b) onto this plane and obtain the projected line segment (P'_a, P'_b).

5. Intersection Point between a Line and a Plane

Given a line segment (P_a, P_b) and a plane with a unit normal $\vec{N}$ and another point P_0 on the plane, we want to find out the intersection point between this line segment and the plane. First, we check the signs of dot products $(P_a - P_0)\vec{N}$ and $(P_b - P_0)\vec{N}$. If any of them is equal to zero, then at least one of the points is on the plane. If both of them have the same sign, the two points P_a and P_b are on the same side of the plane and, therefore, there will be no intersection point. If the signs are different, we calculate the distances from the points P_a and P_b to the plane (say, d_1 and d_2, respectively). The intersection point will be $(d_2 \times P_a + d_1 \times P_b)/(d_1 + d_2)$.

6. Nearest Distance between Two Line Segments in 3D space

Two line segments (P_a, P_b) and (P_c, P_d) are defined in the 3D space. We want to calculate the nearest distance between the two line segments. First, we project the line segment (P_a, P_b) to a plane that is parallel to the line segment (P_a, P_b) and contains the line segment (P_c, P_d) using the method mentioned in Section 4 of the appendix, which will be (P'_a, P'_b). We also calculate the distance between the point P_a and the plane (Section 2 of appendix), which is defined as d_1. We then find the nearest distance (d_2) between the two line segments (P'_a, P'_b) and (P_c, P_d) that are in the same plane. Then, the nearest distance between the two line segments will be the square root of $(d_1 \times d_1 + d_2 \times d_2)$.

7. Movement of a Point Penetrates a Triangular Polygon

A point is at point P_1 in time t_1 and at point P_2 in time t_2. A triangular polygon has vertices P_a, P_b, and P_c. We want to check if the movement of the point penetrates the triangular polygon. First of all, we find the normal $\vec{N}$ of the triangle, which is equal to the cross product of the two vectors $(P_a - P_c)$ and $(P_b - P_c)$ and normalize it. We then check if there is an intersection between the line segment (P_1, P_2) and the plane containing the triangle (Section 5 of appendix). If there is an intersection, we check whether the intersection point is inside the triangle or not. If the intersection point is inside the triangle, the movement of the point penetrates the triangle.

8. Collision between a Moving Line Segment and a Static Line Segment in 3D Space

At time t_0, the line segment l_{ab} is at $l_{ab}(t_0)$, and at time t_1, it moves to $l_{ab}(t_1)$. We want to know if the movement of l_{ab} from t_0 to t_1 passes the line segment l_{cd}. The analytical solution could be found in Schomer and Thiel (1995). Their method gives the exact solution for the collision time and the collision point. However, this solution is computationally too expensive to be used in haptic rendering since the haptic loop needs to be updated at about 1 kHz. Instead, we present a simplified method in here. Although this method does not calculate the exact collision time and the collision point, it reports whether the movement of one line crosses the other. For this method to be valid, the translation and rotation of the line segment should be very small. (This is absolutely the case in haptic rendering since the haptic loop is updated at about 1 kHz and the movements of our hand are quite slow.) To detect the collision, we first calculate the vector $\vec{D}_0$, which represents the nearest distance from l_{cd} to $l_{ab}(t_0)$ (see Section 6 of appendix). Let us call the nearest points on the two lines P_{cd0} and P_{ab0}. We also calculate the nearest distance vector $\vec{D}_1$ from l_{cd} to $l_{ab}(t_1)$. Let's call the nearest points on the two lines P_{cd1} and P_{ab1}. If (1) the dot product of $\vec{D}_0$ and $\vec{D}_1$ is negative, (2) neither P_{cd0} nor P_{cd1} are the end points of l_{cd}, (3) P_{ab0} is not the end point of $l_{ab}(t_0)$, and (4) P_{ab1} is not the end point of $l_{ab}(t_1)$, we say the movement of l_{ab} from t_0 to t_1 crosses the line segment l_{cd}.

Acknowledgments

The research work reported in this paper was supported by a grant from the U.S. Office of Naval Research.

References

Adachi, Y., Kumano, T., and Ogino, K. 1995. Intermediate representation for stiff virtual objects. *Proc. IEEE Virtual Reality Annual Intl. Symposium*, Research Triangle Park, NC, March 11–15, pp. 203–210.

Avila, R. S., and Sobierajski, L. M. 1996. A haptic interaction method for volume visualization. *IEEE Proceedings of Visualization*, pp. 197–204.

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

Basdogan, C., Ho, C., and 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*, Dallas, TX, November 15–21, DSC-61, pp. 77–84.

Basdogan C., Ho, C., Srinivasan, M. A., Small, S., and Dawson, S. 1998. Force interactions in laparoscopic simulations: Haptic rendering of soft tissues. *Proceedings of the Medicine Meets Virtual Reality VI Conference*, San Diego, CA, January 19–22, pp. 385–391.

Buttolo, P., and Hannaford, B. 1995. Pen-based force display for precision manipulation in virtual environments. *Proceedings IEEE Virtual Reality Annual International Symposium*, NC, March, pp. 217–224.

Chen, J., DiMattia, C., Falvo, M., Thiansathaporn, P., Superfine, R., and 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*, Dallas, TX, pp. 167–171.

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

Fritz, J., and Barner, K. 1996. Haptic scientific visualization. *Proceedings of the First Phantom Users Group Workshop*, ed. J. K. Salisbury and M. A. Srinivasan, MIT-AI TR-1596 and RLE TR-612, Dedham, MA.

Gottschalk, S., Lin, M., and Manocha, D. 1996. OBB-Tree: A hierarchical structure for rapid interference detection. *ACM, Proceedings of SIGGRAPH*, New Orleans, LA, August, pp. 171–180.

Green, D. F., and 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*, Dedham, MA, October 19–22.

Gregory, A., Lin, M., Gottschalk, S., and Taylor, R. 1999. A framework for fast and accurate collision detection for haptic interaction. *Proceedings of IEEE Virtual Reality Conference*, pp. 38–45.

Ho, C., Basdogan, C., and Srinivasan, M. A. 1999. Efficient point-based rendering techniques for haptic display of virtual objects. *Presence: Teleoperators and Virtual Environments* 8(5):477–491.

Iwata, H. 1993. Pen-based haptic virtual environment. *Proc. VRAIS IEEE'93*, Seattle, WA, pp. 287–292.

Lin, M. 1993. *Efficient Collision Detection for Animation and Robotics*. Ph.D. thesis, University of California, Berkeley.

Mark, W., Randolph S., Finch, M., Van Verth, J., and Taylor, R. M. 1996. Adding force feedback to graphics systems: Issues and solutions. *Computer Graphics: Proceedings of SIGGRAPH*, New Orleans, LA, August, pp. 447–452.

McNeely, W. A., Puterbaugh K. D., and Troy, J. J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. *Proceedings of SIGGRAPH*, Los Angeles, CA, pp. 401–408.

Millman, P., and Colgate, J. E. 1991. Design of a four degree-freedom force reflection manipulandum with a specified force/torque workspace. *IEEE International Conference on Robotics and Automation*, Sacramento, CA, pp. 1488–1493.

Minsky, M., Ming, O., Steele, F., Brook, F. P., and 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., and Canny, J. 1995. Impulse-based simulation of rigid bodies. *Proceedings of Symposium on Interactive 3D Graphics*, April, pp. 181–188.

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

Ruspini, D. C., Kolarov, K., and Khatib, O. 1997. The haptic display of complex graphical environments. *ACM, Proceedings of SIGGRAPH*, Los Angeles, CA, July, pp. 345–352.

Salcudean, S. E., and 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., and Srinivasan, M. A. 1997. Phantom-based haptic interaction with virtual objects. *IEEE Computer Graphics and Applications* 17(5):6–10.

Salisbury, J. K., and Tarr, C. 1997. Haptic rendering of surfaces defined by implicit functions. *Proceedings of the ASME* DSC-61:61–67.

Schomer, E., and Thiel, C. 1995. Efficient collision detection for moving polyhedra. *ACM, 11th Computational Geometry*, Vancouver, B.C., ACM 0-89791-724-3/95/0006.

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

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

Thompson, T. V., Johnson, D. E., and Cohen, E. 1997. Direct haptic rendering of sculptured models. *Proc. Symp. Interactive 3D Graphics*, Providence, RI, April 27–30, pp. 1–10.

Zilles, C. B., and 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.