Haptic Display for Human Interaction with Virtual Dynamic Environments

Diego Ruspini* **and Oussama Khatib**
Robotics Laboratory
Department of Computer Science
Stanford University
Stanford, CA 94305-9010

Received 16 August 2001; accepted 16 August 2001

Haptics is an emerging technology that permits direct "hands-on" interaction with a virtual environment. A haptic device uses mechanical actuators to physically push a user's finger or hand to give the sensation that he or she would have when interacting with a real physical environment. These force feedback systems have many applications, from training a surgeon to perform an operation, to assisting a child in understanding the behavior of a lever or pulley. In this paper we discuss methods and techniques to allow realistic and robust haptic interactions between a human and a complex dynamic virtual environment. Beyond modeling object penetration constraints, this work demonstrates how shading, friction, texture, and dynamics can be generated to create compelling and realistic virtual worlds.

1. INTRODUCTION

Haptic systems have been around for a number of years. Early haptic rendering systems modeled surface contacts by generating a repulsive force proportional to the amount of penetration into an obstacle. While these penalty-based methods worked well to model simple obstacles, such as planes or spheres, a number of difficulties are encountered when trying to extend these models to display more complex environments.

*To whom all correspondence should be addressed; e-mail: ruspini@cs.stanford.edu.

When multiple primitives touch or are allowed to intersect it is often difficult to determine what the appropriate restoration force should be. Simply adding the penetration force from each object can result in the creation of large forces that could potentially cause damage to the haptic device or injury to the user. In addition, the penetration distance and direction into an obstacle is not always uniquely defined. As a user presses into an obstacle, at some point the user's position will be nearer to a surface other than the surface of original penetration. When this "pop-through" occurs, the user will be actively pushed through the object, resulting in an unrealistic and usually undesirable

sensation. Lastly, small or thin objects may not have a sufficient internal volume to create the constraint forces required to prevent the probe from passing through the obstacle.

To be useful in building real-world applications, a haptic rendering system must be capable of displaying the models found in common graphic applications. Typical graphic models consist of a large number of points, lines, and polygons that define the boundary surface of the object. Often described as "polygon soups," these representations often contain intersecting primitives and gaps between polygons that are intended to represent a connected topological surface. Penalty-based methods appear to be ill-suited in modeling these types of environments.

We propose an alternative method for computing the restoring forces required to simulate contact with virtual models, which does not depend on determining the penetration distance into an obstacle. In our method, a representative object, a "proxy," substitutes in the virtual environment for the physical finger or probe. This "virtual proxy" can be viewed as if connected to the user's real finger by a stiff spring. As the user moves their finger in the workspace of the haptic device, they may pass into or through one or more of the virtual obstacles. The proxy, however, is stopped by the obstacles and quickly moves to a position that minimizes its distance to the user's finger position (Figure 1). The haptic device is used to generate the forces of the virtual spring, which appear to the user as the constraint forces caused by contact with a real environment. This approach is similar to the method first proposed by Zilles and Salisbury,[36] but does not require that the topology of the surface be computed a priori, making it more applicable to graphical environments and scene graphs containing moving or intersecting obstacles. In addition, we extend the constraint-based approach to correctly and robustly utilize additional graphical information such as shading normals, friction, and texture.

The remainder of this paper is organized as follows: In Section 2, the update procedure for the proxy position is presented. In Section 3, a method for simulated smooth curved surfaces is discussed. Methods to render static, dynamic, and viscous friction are described in Section 4. Texture is introduced in Section 4.3. Section 5 describes the performance of fast collision detection to maintain the high update rates required for haptic display. A short summary of how dynamic models are introduced into the haptic simulation, and an overview of the system, are given in Sections 6 and 7. The stability and control of the haptic system is detailed in Section 8. Lastly, Sections 9 and 10 describe some of the potential applications, show some examples, and present the conclusion.

2. UPDATING THE PROXY POSITION

From an algorithmic point of view, it can be easily seen that the motion of the proxy is very similar to that of a robot reactively moving towards a goal (the user's finger) under the influence of an artificial potential field.[18] When unobstructed, the proxy moves directly towards the goal. If the proxy encounters an obstacle, direct motion is not possible, but the proxy may still be able to reduce the distance to the goal by moving along one or more of the contact surfaces. When the

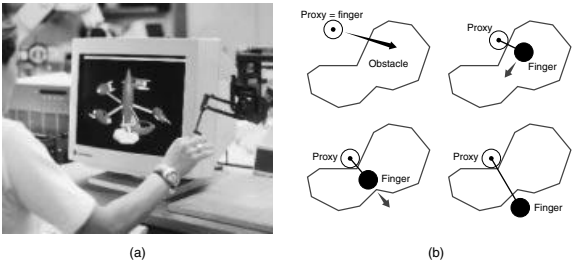

Figure 1. (a) A user is haptically interacting with a dynamic virtual environment. The sensation of contact is created by applying forces through the haptic device to move the user's finger to the location of the constrained representative object (the proxy). (b) The virtual proxy moves to locally minimize the distance to the user's finger position subject to the constraints in the environment.

proxy is unable to further decrease its distance to the goal, it stops at the local minimum configuration.

As we will see, many of the algorithms and methods developed for this haptic rendering method are derivatives of methods originally proposed for robotic applications. For this discussion, we will model the proxy as a smooth massless sphere. The radius of the proxy is selected to be large enough to be easily visible for graphic display, and to prevent it from falling through small gaps that may exist between the polygonal patches representing the surface of the obstacle. These gaps are common in graphic models, and repairing a model to eliminate such "leaks" is, in general, prohibitively expensive, especially for interactive applications.

An iterative solution is used to find the local minimum energy configuration of the proxy. Each iteration is divided into two stages: *move* and *update*.

2.1. Moving the Proxy

In the move stage, the volume swept by the virtual proxy, as it moves along a linear path towards its goal, is checked to see if it penetrates any of the primitives in the environment. The initial goal configuration is the user's finger position, but this will change as dictated in the update stage on subsequent interactions. If the proxy's path does not collide with any obstacles, the proxy is moved directly to the goal. Otherwise, the proxy is moved until it makes contact with the first primitive or primitives along the path.

A naive comparison of the proxy's path with each primitive in the environment would be unable to achieve a sufficiently high update rate (for haptic display) on any but the most simple models. A common technique to reduce the number of low-level comparisons that must be made is to surround each obstacle with a hierarchy of bounding volumes. In our system, a bounding sphere hierarchy is used (described in detail in Section 5). Many other methods of high-level collision detection have also been proposed.[14] All these techniques exploit spatial or temporal coherence to quickly eliminate from the environment primitives that lie too far from the proxy's path to affect its motion. Primitives that are not eliminated by the high-level pruning technique must be checked individually to see if they intersect the proxy's path. This can be accomplished efficiently with algorithm's such as Gilbert's[12] or Lin-Canny,[21] which can quickly compute the distance between two convex polyhedra. The low-level test is also similar to the ray intersection test used in ray-tracing applications for graphics. The Gilbert algorithm[12] finds the distance between the convex hull of two sets of points $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$. On completion, the algorithm will return a set of weights $w = \{w_1, \ldots, w_{\max(n,m)}\}$ such that

$$p_{\text{nearest}} = \sum_{i=1}^{\max(n,m)} p_i w_i, \quad q_{\text{nearest}} = \sum_{i=1}^{\max(n,m)} q_i w_i, \quad \sum_{i=1}^{\max(n,m)} w_i = 1, \quad w_i \geq 0 \tag{1}$$

Gilbert's algorithm was selected for our implementation because it requires little preprocessing compared to other methods and can be used in other parts of the haptic rendering process (as will be shown).

Once the point of contact has been found, the proxy's position is updated to this new configuration. At this point, direct movement to the goal is no longer possible, but it may still be possible to reduce the distance to the user's finger position. In the update phase this new goal configuration is found.

2.2. Updating the Goal Position

If the proxy is currently at the user's finger position, no further work is required. In general, however, the proxy will not be exactly at the goal configuration and a new direction, constrained by the contact surfaces, must be found to further decrease the distance to the goal. The available free space around the proxy can be effectively modeled by examining its configuration space.[20] In this space, the proxy is represented as a point identifying the center of the proxy. The primitives in contact with the proxy are mapped to *configuration space obstacles* (C-obstacles), consisting of all points within one proxy radius of the original obstacles. For each primitive, a unique constraint plane tangent to the configuration space surface and going through the proxy position can be defined. Locally, each constraint plane limits the potential motion of the proxy to the half-space above the plane.

The intersection of all such half-spaces defines a convex, unbounded polyhedron that represents locally all points reachable by direct linear motion from the current proxy position. The new goal configuration is the point in the free-space polyhedron nearest to the user's finger position. The user's finger position will always be situated beneath all the constraint planes. An example of the configuration space, C-obstacles, and proxy constraint planes is shown in Figure 2.

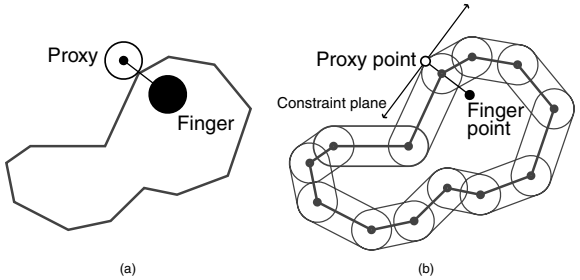

Figure 2. (a) Actual and (b) configuration space obstacles.

The problem is to find the desired goal configuration x:

$$\begin{aligned} \text{minimize} \quad & D(x) = \|x-u\| \quad \text{such that} \\ & \hat{n}_1^T x \geq d_1 \\ & \hat{n}_2^T x \geq d_2 \\ & \vdots \\ & \hat{n}_m^T x \geq d_m \end{aligned} \tag{2}$$

where u is the current position of the user's finger, and $\hat{n}_i^T x = d_i$, $0 \leq i \leq m$ are the equations for the m constraint planes in contact with the proxy. Here we will assume that all $\hat{n}_i$ are unit normals.

The solution can be seen as finding the point x on the free-space polyhedron nearest to the user's position u. While the user's position can be treated as a degenerate polyhedron consisting of only one point, the specification of our free-space polyhedron is not in a form that can be directly exploited by the Gilbert algorithm. The free-space polyhedron is unbounded and the limits of its extent are defined by the intersection of all the constraint planes, not vertices. There does exist, however, a dual relationship between this free-space polyhedron and another polyhedron that does satisfy the requirements for the use of the algorithm.

Without loss of generality, we will assume that the proxy is centered at the origin and that the user's finger position is located at a point one unit away from the proxy position. The frame of reference and unit of measure can be changed if this is not the case. All the constraint planes go through a common point (the proxy position = the origin) and can therefore be defined uniquely by their surface normals $\hat{n}_i^T x = 0$, $0 \leq i \leq m$.

In the 3D case, the proxy's position can be locally constrained by, at most, three constraint surfaces; all other contact surfaces can be considered redundant. The planes associated with these constraint surfaces will be referred to as the set of *active* planes. All other constraint planes will be considered *inactive*. At first, we will assume that an oracle has identified, out of all the original planes, which planes will belong to the active set. We will designate these planes by their surface normals n_{a_i}, $0 \leq i \leq m' \leq 3$. Later we will see that the oracle is not required.

Considering only the active planes and rewriting equation 2 in matrix notation, the constraint problem can be written as

$$\begin{aligned} \text{minimize} \quad & D(x) = \frac{1}{2}(x-u)^T(x-u) \quad \text{such that} \\ & N^T x = 0 \end{aligned} \tag{3}$$

where $N = [\hat{n}_{a_1} \ldots \hat{n}_{a_{m'}}]$, $0 \leq m' \leq 3$ is the matrix containing normals of the active constraint planes. Introducing the m' multipliers $\lambda = [\lambda_1 \ldots \lambda_{m'}]$, the Lagrangian for Eq. 3 becomes

$$L = \frac{1}{2}(x-u)^T(x-u) + \lambda^T(N^T x) \tag{4}$$

The minimum configuration is found where the derivatives of L are zero. Taking the partial with respect to x, we obtain

$$\partial L/\partial x = (x-u) + N\lambda = 0 \tag{5}$$

Solving for x, we obtain a relation between the solution x and the Lagrange multipliers λ.

$$x = u - N\lambda \tag{6}$$

Substituting for x in Eq. 4, we obtain a dual for our original constraint equation 3:

$$\begin{aligned}
&\text{maximize} \\
&-P'(x) \\
&= \frac{1}{2}(-N\lambda)^T(-N\lambda) + \lambda^T N^T(u - N\lambda) \\
&= \frac{1}{2}\lambda^T N^T N\lambda + \lambda^T N^T u - \lambda^T N^T N\lambda \\
&= -\frac{1}{2}\lambda^T N^T N\lambda + \lambda^T N^T u \\
&= -\frac{1}{2}(\lambda^T N^T N\lambda - 2\lambda^T N^T u + u^T u) + \frac{1}{2}u^T u \\
&= -\frac{1}{2}(N\lambda - u)^T(N\lambda - u) + \frac{1}{2}u^T u.
\end{aligned} \tag{7}$$

Rewriting Eq. 7 as a minimization, and noting that u is a unit normal ($u^T u = 1$), the dual solution can be found to be equivalent to the solution of

$$\text{minimize} \quad P(x) = \|N\lambda - u\| \tag{8}$$

In this equation, $P(x)$ can be thought of as representing the potential energy of the system. The solution x (that minimizes the distance to the user's position) is the configuration that minimizes the potential energy stored in the virtual spring that exists between the user and the proxy.

To solve this equation using Gilbert's algorithm, we will at first make the following substitutions. First we will define a space S' such that

$$S' = \begin{bmatrix} 0 \\ 0 \,\big|c_{a_1}\hat{n}_{a_1}\big| \cdots \big|c_{a_{m'}}\hat{n}_{a_{m'}}\big| \\ 0 \end{bmatrix} \tag{9}$$

where $c_{a_i}, 0 \le i \le m'$ is a constant, such that

$$c_i = \frac{1}{u^T \hat{n}_i} \tag{10}$$

Next, we can define a set of $m'+1$ weights w' such that

$$w' = \left[w_0 \left| \frac{1}{c_{a_1}}\lambda_0 \right| \cdots \left| \frac{1}{c_{a_{m'}}}\lambda_{m'} \right| \right]^T \tag{11}$$

where $\lambda_i, 0 \le i \le m'$ are the terms from the vector λ and w_0 is a new constraint variable whose value we define later.

Recalling that $N = [\hat{n}_{a_1} \ldots \hat{n}_{a_{m'}}]$, it is trivial to show that $S'w' = N\lambda$, and Eq. 8 can now be rewritten as

$$\text{minimize} \quad P(x) = \|S'w' - u\| \tag{12}$$

The solution to Eq. 12 is the point nearest to u in the space spanned by $S'w'$. If the solution we desire is contained in the convex hull created by the columns of S', then the solution can be found by invoking Gilbert's algorithm. To prove that the solution lies in this space, we must show that $\sum_{0 \le i \le m'} w_i = 1$ and $w_i \ge 0, \forall w_i, 0 \le i \le m'$ where w_i is the ith term of vector w.

From the original problem statement, $x_p = [0\ 0\ 0]^T$ satisfies the constraints of the system (it is the current valid proxy position) with $D(x_p) = 1$. Given that the solution x must be nearer or at least the same distance as the current configuration, we see that $(u-x)^T(u-x) \le \|u - x\| \le \|u - x_p\| \le 1$. Noting from Eq. 6 that $u - x = N\lambda$, and our original requirement that planes on the active set satisfy $n_i x = 0$, we see that

$$\begin{aligned}
&(u-x)^T(u-x) \\
&= (u-x)^T(N\lambda) \\
&= (u-x)^T(S'w') \\
&= (u-x)^T\left(0w_0 + \frac{1}{u^T\hat{n}_{a_1}}\hat{n}_{a_1}w_1 + \cdots \right. \\
&\quad \left. + \frac{1}{u^T\hat{n}_{a_{m'}}}\hat{n}_{a_{m'}}w_{m'}\right) \\
&= \left(\frac{(u-x)^T\hat{n}_{a_1}}{u^T\hat{n}_{a_1}}\right)w_1 + \cdots + \left(\frac{(u-x)^T\hat{n}_{a_{m'}}}{u^T\hat{n}_{a_{m'}}}\right)w_{m'} \\
&= \left(\frac{u^T\hat{n}_{a_1} - x^T\hat{n}_{a_1}}{u^T\hat{n}_{a_1}}\right)w_1 + \cdots \\
&\quad + \left(\frac{u^T\hat{n}_{a_{m'}} - x^T\hat{n}_{a_{m'}}}{u^T\hat{n}_{a_{m'}}}\right)w_{m'} \\
&= \left(\frac{u^T\hat{n}_{a_1} - 0}{u^T\hat{n}_{a_1}}\right)w_1 + \cdots + \left(\frac{u^T\hat{n}_{a_{m'}} - 0}{u^T\hat{n}_{a_{m'}}}\right)w_{m'} \\
&= (1)w_1 + \cdots + (1)w_{m'} \\
&= w_1 + \cdots + w_{m'} \le 1
\end{aligned} \tag{13}$$

Thus, the first constraint is satisfied by setting the unconstrained variable w_0 equal to

$$w_0 = 1 - \sum_{i=1}^{m'} w_i \tag{14}$$

While it is trivial to show that $w_0 > 0$, proving nonnegativity for the other elements of w is more

problematic. The normals of the constraint planes may be redundant, permitting an infinite number of solutions, some of which may have negative weights. However, only the minimal set of planes for which all the weights are positive will be considered as candidate active sets. To see the reason for this distinction, note that the force exerted by the virtual spring on the proxy is given by the equation:

$$f = k_s(u - x) = k_s(N\lambda) \tag{15}$$

where k_s is some positive spring constant. The individual force applied by a given constraint plane to oppose the motion of the proxy is given by

$$f_i = -k_s(\hat{n}_i \lambda_i) \tag{16}$$

Each constraint surface can only push, not pull, on the proxy. Therefore, the force normal to the surface must be nonnegative ($n_i^T f_i = -k_s\lambda_i = -k_s c_i w_i \geq 0$). As $u^t n_i \leq 0$ (the user's position is below the constraint plane by definition), we see that $c_i \leq 0$ and therefore $w_i \geq 0$.

Having shown that the desired solution lies in the convex hull of the space defined by columns of S', it is now possible to extend the result so that we no longer require prior knowledge of which constraint planes belong to the active set. The intersection of all the halfspaces defined by the constraint planes of the original problem is represented in the dual by the union of the convex hull for all possible sets of planes. The active set is defined by the polytope whose distance is the smallest to the user's position.

We now have a means to find the new goal position efficiently. Eliminating our assumption that the finger position is a unit length away from the current proxy position, let $\hat{u} = u/\|u\|$. We use Gilbert's algorithm to find the nearest point between $\hat{u}$ and a polyhedra defined by the $m+1$ points that form the columns of the dual space:

$$S = \begin{bmatrix} 0 \\ 0 \,|\, c_1\hat{n}_1 \,|\, \cdots \,|\, c_m\hat{n}_m \\ 0 \end{bmatrix} \tag{17}$$

The goal displacement can be found by

$$x = \|u\|(u - Sw) \tag{18}$$

where w is the vector of weights that are returned by the distance algorithm. The nonzero elements of w define that set of active constraints. The displacement x can be added to the current proxy position to define the next goal configuration. In addition, the force applied by the user on each constraint plane can be found by Eq. 16, after scaling, to be

$$f_i = (k_s\|u\|c_i w_i)\hat{n}_i \tag{19}$$

An example of this dual relation is illustrated in Figure 3. In this example configuration, the proxy position is constrained by two constraint planes $\hat{n}_1$ and $\hat{n}_2$. These constraints map to a dual-space triangle defined by the origin and the points along the negative normal directions of $\hat{n}_1$ and $\hat{n}_2$. As can be see in the illustration, the distance P closest to the finger position direction $\hat{u}$ is proportional to the distance that the goal configuration is away from the current proxy position. As the finger is moved around the proxy position, the

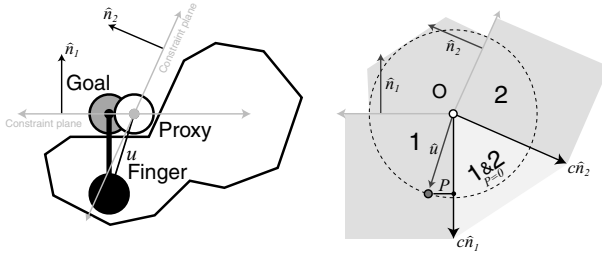

Figure 3. (a) The new goal configuration is selected as the point in the free-space nearest the user's finger position. (b) Its equivalent dual-space representation represents the change in potential caused by moving from the current configuration.

relationship is maintained. When the user's finger is inside the triangle, the distance to the hull is zero and corresponding to the configurations where the proxy is completely constrained.

3. HAPTIC SHADING

As described, the proxy's position is selected to minimize its distance to the user's finger position, subject to the constraints in the environment. In many cases, however, the movement of the proxy can be altered to create a variety of other useful haptic effects. An alternative minimization is to use information found in many graphic models to allow regular polygonal surfaces to be perceived as if they were constructed out of curved continuous surfaces. As illustrated in Figure 4, surface normals are defined at the vertices of a polygonal mesh that correspond to the surface normals of an underlying curved surface. To draw a given polygon, the graphics hardware interpolates the normals[27] or a corresponding color value[15] for each pixel on the surface. The lighting calculations are performed using the interpolated surface normal information instead of the surface normal of the polygon. This has the effect of eliminating abrupt surface color changes between polygon boundaries and giving the appearance of a curved continuous surface. The drawn surface is however still composed of individual polygonal surfaces allowing fast graphic rendering on dedicated hardware.

For haptics, these given vertex surface normals can be used to give the sensation that the user is touching a continuous, nonfaceted surface. This modeling makes use of a haptic illusion first described by Minsky,[24] who was able to haptically display three-dimensional height fields on a two-degree-of-freedom planar haptic display. While unable to apply forces in the z (height) direction, the illusion of a three-dimensional surface was created by applying tangential forces proportional to the slope of the field. The basis for the illusion derives from the disparity between human force and position differentiation capabilities. While humans are able to distinguish small force changes, they are relatively incapable of noticing small position disparities. By combining generated tangential forces with the normal force created by the physical constraints (in z) of the 2D haptic device, an appropriate contact force can be applied to the user's finger. The position of the user's hand in z, however, remains fixed.

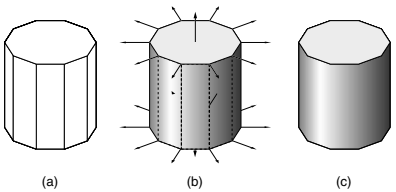

Figure 4. (a) In most graphic systems, curved objects are modeled as a set of flat polygonal patches. (b) To achieve the appearance of a continuous surface, surface normals are defined on the vertices of each patch. (c) The color or lighting models are interpolated over the surface to produce a continuously shaded surface.

Morgenbesser[25] and Srinivasan[34] were the first to try to use this illusion to shade virtual models. In their solution, the direction of the normal force is changed while retaining the magnitude caused by the penetration of the original object. Their work, however, required that the topology of the surface be known, limiting applicability when the environment contained intersecting or moving obstacles. In addition, contact with multiple constraint surfaces was not considered. Such a case would occur if a user, for example, were following the crevasse created around the contact region of two side-by-side shaded cylinders. In our approach, rather than directly adding or altering forces applied to the user, an alternative minimization is used to determine the best goal position for the proxy. Since the approach only alters the position of the proxy, and not directly the forces applied to the user, stable performance is much easier to guarantee.

When contact occurs with a polygonal surface containing vertex-defined normals, a new local surface normal is calculated by interpolating the normals from the vertices of the polygon. This process is very similar to the interpolation done in graphics, but has a few caveats (which are discussed in Section 3.1). Once the interpolated normal is known, it can be used to define a constraint plane going through the current proxy position.

The haptic shading method proceeds in two passes. In the first pass, the new goal solution is found as in the update phase described in Section 2.2. In this pass, however, the interpolated constraint plane is used instead of the original for any contact surface containing user-supplied normals. This new subgoal can be thought of as the desired goal configuration of the underlying curved model. This goal position may, however, violate the constraints of the original polygonal geometry since it may lie above or below the true object surface. Instead, the update procedure

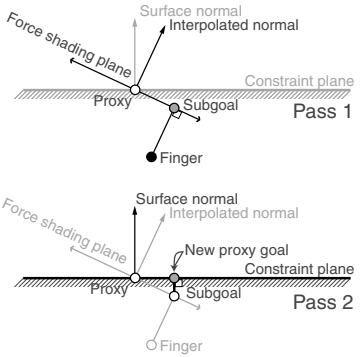

Figure 5. Two pass haptic shading with specified normals.

is called again but with the original (noninterpolated) constraint planes, and substituting the goal configuration generated in the first pass for the user's finger position. This two-pass approach has the effect of finding the nearest valid configuration to the minimal configuration as defined by the interpolated surface normals.

An example of this approach is shown in Figure 5. Note that after the first pass the goal position lies below the surface of the object. After the second pass a valid proxy goal on the surface of the original obstacle is found. This goal is to the right of the goal position that would have been found if shading were not applied. If during the next move phase no obstacle is encountered, the proxy will move to this configuration and a force pulling the user's finger to the right will be applied, as would be expected from an object having the surface normal illustrated.

If the subgoal configuration after the first pass is above all the true constraint planes, the subgoal is first projected back onto the nearest true constraint plane. This ensures that the new subgoal will always be on the object surface, and that surface effects like friction and texture will be handled correctly.

The difference between a haptically shaded surface, a flat surface, and the true curved surface is illustrated in Figure 6. In all the figures, the differences between the user's positions and the positions of the proxy are shown as the user's finger follows a circular counterclockwise path around the object. As seen in Figure 6(a), a strong discontinuity occurs when the proxy reaches each edge of this ten-sided polygonal approximation of a circular obstacle. This results in a force discontinuity that gives the user the impression of crossing over an edge. In Figure 6(b), surface normals have been specified on vertices of the obstacle. The resulting movement of the proxy shows that the resultant force is always perpendicular to the interpolated surface just as in the case of a true circular object. The effect of this minimization is to eliminate the large instantaneous changes in force that normally

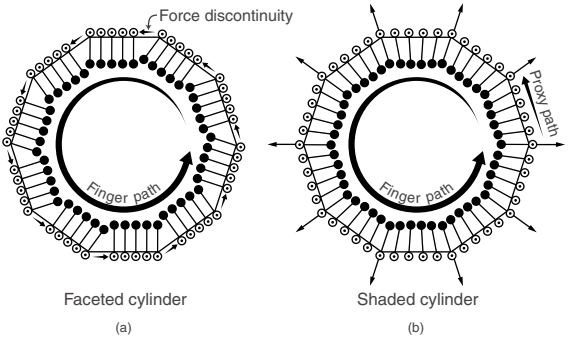

Figure 6. Haptic shading (b) eliminates the force discontinuities associated with moving along a faceted cylindrical surface (a). Although the path of the finger and proxy differ from that of a true cylinder (c), human position discrimination ability is insufficient to distinguish the tactile differences between the two displays.

occur at polygon boundaries, resulting in a surface that feels smooth and continuous. The discrimination abilities of humans are insufficient to detect the small positional differences between the polygonal and underlying curved surface.

3.1. Determining Shaded Surface Normals

For the previously described shading algorithm, the desired surface normal for the shading constraint plane must be found by interpolating its value from the normals defined on the vertices of the primitive. When the contact point is on the surface of the polygon, the weights used for the interpolation can be obtained from the collision-detection algorithm. Gilbert's distance algorithm returns the nearest point on the surface as a weighed sum of a set of vertices on the polygon. These same weights can be used to find the shading normal; see ref. 12 for more information. As the contact point may lie on either side of the polygonal primitive, a check should be made to ensure that the interpolated surface normal points away from the obstacle. The interpolated normal $\hat{n}$ should be inverted if $\hat{n}' \cdot \hat{n} > 0$, where $\hat{n}$ is the outward normal of the original primitive.

While finding the interpolated normal for surface contact is fairly straightforward, special consideration must be given if the proxy is in contact with one of the edges or vertices of the polygon. As is illustrated in Figure 7, the shaded constraint surface is found on the configuration space obstacle and not on the original primitive. On the surface, the interpolated surface normal can be mapped to the top and bottom surfaces of the configuration space obstacles, as illustrated in Figure 7(b). It is unclear, however, what mapping should be used for points on the surface outside this region. Extrapolating the surface normals will result in boundary values that depend on all the vertices of the polygon, making it difficult to create patches that will form a single continuous surface when placed together. Using the same surface values as the nearest edge or vertex will lead to large differences between the interpolated and true surface normal, and will have a singularity where the top and bottom surface meet. Interpolating around the angle formed by the edge will also result in interfering shading planes if the edge is shared by two shaded polygons representing a continuous surface.

In our approach, if contact is made with the boundary of a configuration space obstacle, a hypothetical configuration space surface is created that is tilted so that its upper and lower surface match those of the nearest interpolated edge or vertex normals. An example of this is illustrated in Figure 7(a). The configuration space normals of this hypothetical surface are projected onto the actual configuration space boundary to define the shaded surface normals for the configuration space obstacle. Some example configurations of shaded normals are illustrated in Figure 7(c). This mapping has an important property in that patches, which share a common edge and have identical surface normals defined on the vertices of this edge, will feel smooth to the user.

4. SURFACE PROPERTIES

Several researchers have proposed methods to simulate static, dynamic, and viscous friction, and texture.[2,6,22,32,33] These methods worked by introducing additional force to simulate the forces of friction from

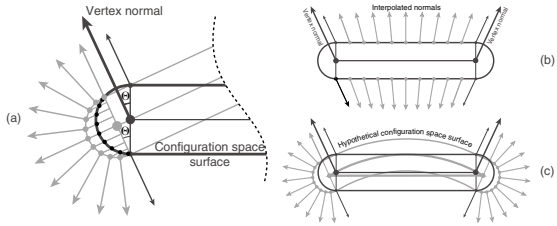

Figure 7. The surface normals of a hypothetical configuration space obstacle are projected onto the true configuration space surface, to define the shading normals used to simulate a curved surface.

the contact surface, and often depended on estimates of the finger's velocity that made stability of the solution very difficult to guarantee. All these effects, however, can be created by restricting or changing the motion of the proxy. This results in a haptic controller that is much more stable and easier to control.

4.1. Static Friction

Static friction (stiction) is particularly simple to model within the virtual proxy framework. The force exerted on the proxy by the user can be estimated by the equation $f = k_p(p - v)$, where p is the position of the proxy, v is the position of the finger, and k_p is the proportional gain of the haptic controller. For a given constraint plane, let f_n and f_t be the components of the force on the proxy normal and tangential to the constraint plane, respectively. If the given constraint surface has a static friction parameter μ_s, then the proxy is in static contact if $\| f_t \| \leq \| f_n \|$, that is, the user's position is in the friction cone of the surface. An example of such a configuration is shown in Figure 8(a). When any constraint surface is in static contact with the proxy, the proxy's position is prevented from changing by making the new subgoal position equal to the current proxy position.

4.2. Viscous and Dynamic Friction

Viscous and dynamic friction can be modeled by looking at a simplified set of equations for the motion of the proxy. As illustrated in Figure 8(b) the equations of motion for the proxy can be written as:

$$m\ddot{x} + b\dot{x} = f_{\text{finger}} + f_N - \mu f_N \tag{20}$$

where x is the position of the proxy, m is its mass, b is the viscous damping term, and f_{finger}, f_N, and $-\mu f_N$ are the force on the proxy created by the user's finger, the surface constraint, and the drag caused by dynamic friction, respectively. Because the mass of the proxy can be considered as being very small, Eq. 20 can be observed as $m \to 0$. When the mass of the proxy is zero, the body quickly reaches its saturation velocity. In dynamic equilibrium, the velocity of the proxy is given by

$$\dot{x} = \frac{f_{\text{finger}} + f_N - \mu f_N}{b} \tag{21}$$

This limit can be used to bound the amount that the proxy can travel in one clock cycle. When multiple constraint surfaces exist, the lowest-velocity bound is taken as the limit of the proxy's movement. In the event that the maximum velocity is negative, then the dynamic friction term is sufficient to resist all movement and the proxy's position is not changed. If $b = 0$, no viscous term exists and the maximum velocity is not bounded. Since this approach does not require the estimation of the user's finger velocity from a finite set of encoder values, this approach is not susceptible to the errors found in other approaches.

4.3. Texture

Image-mapped texture is often used in graphics to create richer, more realistic environments. As with graphics, texture can be applied to create higher-fidelity scenes than can be realistically created using polygonal surfaces alone. In our system, an image-based texture map can be used to modulate any of the surface properties described in Section 4. In addition, the force-shaded constraint planes can be modified in a manner similar to the bump mapping introduced by Blinn[5] for computer graphics. The contact-point weights, used for shading, are used to interpolate a texture coordinate from coordinates defined on the vertices of the polygon. The texture coordinates map to an image-based texture, and are converted to displacements to the surface original or shaded normal as in Blinn. Once the texture normal is found, it can be used as in the case of shading in Section 3. An

Figure 8. (a) Static friction can be simulated by not permitting proxy movement if the user's finger is in a given friction cone. (b) Viscous and dynamic friction can be modeled by constraining the motion of the proxy subject to the forces applied to it.

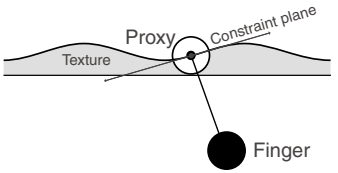

Figure 9. An image-based texture is used to alter the constraint plane to create the sensation of bumps on the virtual surface.

example of this is illustrated in Figure 9. As with shading, the texture does not lift the proxy off the surface of the original obstacle but displaces the goal tangential to the configuration space surface.

Unlike bump mapping for computer graphics, it may be desirable to allow multiple constraint planes to be specified at a given location. An example would be a grooved or craggy surface where the proxy is in contact with multiple surfaces at the same time. In our current implementation, we allow the user to define multiple texture images for a given surface. One constraint plane is created for each image. The selection of appropriate textures to model a given surface is left to the user.

5. COLLISION DETECTION

Because the environment is normally constructed from a large number of primitives, a naive test that checked whether each primitive was in the path of the proxy would be prohibitively expensive. Instead, a hierarchical bounding representation for the object is constructed to take advantage of the spatial coherence inherent in the environment. The bounding representation, based on spheres, is similar to that first proposed by Quinlan.[28] This hierarchy of bounding spheres is constructed by first covering each polygon with small spheres in a manner similar to scan conversion in computer graphics. These spheres are the leaves of an approximately balanced binary tree. Each node of this tree represents a single sphere that completely contains all the leaves of its descendants.

After covering the object, a divide-and-conquer strategy is used to build the interior nodes of the tree. This algorithm works in a manner similar to quicksort. First, an axis-aligned bounding box that contains all the leaf spheres is found. The leaf spheres are then divided along the plane through the midpoint of the longest axis of the bounding box. Each of the resulting two subsets should be compact and contain an approximately equal number of leaf spheres. The bounding tree is constructed by recursively invoking the algorithm on each subset, and then creating a new node with the two subtrees as children. A cut-away view showing the leaf nodes and bounding sphere hierarchy for a typical model is illustrated in Figure 10. Note that a node is not required to fully contain all the descendant internal nodes, only the descendant leaf nodes.

Two heuristics are used to compute the bounding sphere of a given node. The first heuristic finds the smallest bounding sphere that contains the spheres of its two children. The second method directly examines the leaf spheres. The center is taken as the midpoint of the bounding box already computed. The radius is taken to be just large enough to contain all the descendant leaf nodes. The method that generates the sphere with the smallest radius is used for the given node. The first heuristic tends to work better near the leaves of the tree, while the second method produces better results closer to the root. This algorithm has an expected $O(n \log n)$ execution time, where n is the number of leaf spheres. Once constructed, the time required to determine which primitives may lie in the proxy's path is only $O(\log n)$.

The sphere hierarchy is used to prune the number of low-level checks that must be performed but is not used to determine the exact contact point. If the proxy's path intersects one of the leaf nodes of the hierarchy, then the primitive attached to that leaf is checked to see if it intersects the path of the proxy. In our system, we currently use a very fast distance algorithm by Gilbert, Johnson, and Keerthi.[12] The

Figure 10. Bounding sphere hierarchy of a cat model.

algorithm can quickly find the nearest point between two arbitrary bounded convex polyhedra. No preprocessing of the polyhedra is required. Each primitive is defined by the set of vertices whose convex hull defines the interior of the polyhedron. The nearest point between two polyhedra is returned as the weighted sum of a set of vertices in the polyhedra. These weights are used when shading or texture effects are required.

A cache is maintained to avoid calling the low-level check multiple times for the same primitive during the same iteration. This is possible since several leaf nodes may cover a single primitive. In addition, some spatial coherence information used by the Gilbert algorithm is kept in the cache to reduce the computation time between successive calls to the distance algorithm. The low-level distance check runs in linear time with the number of vertices in the two polyhedra. Since the proxy path is a line segment and the number of vertices in a polygon is typically small (3 or 4), this low-level check can be said to take $O(1)$ constant time.

6. DYNAMICS

Our previous discussion has been limited to the rendering of static environments. To create an engaging virtual world, the user must be able to manipulate and dynamically interact with the virtual objects. In general, a mechanical system can be described by a configuration space vector $q = [q_1 \dots q_n]^T$, where n is the number of dof of the system. The forward-dynamics equations of motion of such a system can be used to obtained the configuration space accelerations of the system. These equations have a general from that can be written as:

$$\ddot{q} = M(q)^{-1}(\Gamma - b(q, \dot{q}) - g(q)) \tag{22}$$

where $M(q)$ is the mass matrix, $b(q, q)$ the centrifugal coriolis vector, $g(q)$ the gravity force vector, and Γ the vector representing the internal and external torques applied to the system through either internal actuation or external forces applied by the environment. A simplified set of equations that neglects the centrifugal coriolis and some dynamic coupling terms is used in our current system. We are in the process of converting the system to make use of an extremely fast variation of Featherstone's articulated body solution[9] to permit the simulation of very complex dynamic models at real-time rates.[7]

When a collision occurs between the proxy and one or more objects in the environment, a force is applied to the user simulating a contact with the surface. In a dynamic environment, an equal and opposite force f_c is applied at the contact point(s), which may induce accelerations on the virtual system. The corresponding joint torque vector is given by

$$\Gamma_{\text{ext}} = J_c^T f_c \tag{23}$$

where J_c is the Jacobian of the contact point c, such that the velocity v of the contact point is given by $v = J_c \dot{\Theta}$. For a simple PD controller, the force applied to the environment can be given by

$$f_c = k_p(p_{\text{proxy}} - p_{\text{finger}}) \tag{24}$$

where k_p is the positional gain, and p_{proxy} and p_{finger} are the current positions of the proxy and finger, respectively. Note that this force is in general not sufficient to prevent penetration between the proxy and objects in the environment, as this equation does not incorporate the internal constraints of the proxy or other objects. A more complete solution to computing the contact forces for rigid body simulation can be found in ref. 31. This simplified model, however, is sufficient for simulating the interactions found in most haptic environments. Once the joint-space accelerations are known, the equations of motion for the system can be integrated, from a given initial joint-space configuration and velocity, to obtain the motion for the entire system over time.

7. SYSTEM OVERVIEW

Our current system runs on two computers: the haptic server and the application client. The separation of the haptic and application/graphic processes was first proposed by Adachi, Kumano, and Ogino[2] to ensure the real-time performance of the haptic control loop. This is important since the haptic servo loop must run at a very high rate, typically greater than 1000 Hz, to achieve a high-fidelity force display. Most application programs typically run at a much slower rate (30 Hz).

The application program communicates to the haptic server through a network interface library HL. The current library supports a limited set of the functions provided for graphics by the GL library for SGI IRIX machines. The bulk of the haptic rendering effort is accomplished by the haptic server, freeing the client machine to perform the tasks required of the user's application. High-level models sent by the client are reorganized, and a bounding sphere hierarchy for each

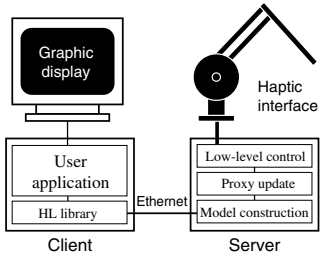

Figure 11. System architecture.

object is constructed. An old model is not replaced until the new model has been sent and is ready for display. Independent of the model construction, the proxy update loop continues to attempt to move the proxy to its minimal configuration. At the lowest level, a real-time controller is used to attempt to move the user's finger to the current proxy position. A diagram of the system is illustrated in Figure 11.

In tests, and in the following examples, the client computer was an SGI Indigo2 running IRIX 6.2. The server was a 200 MHz Pentium Pro running Linux 2.0.2. Communication between computers was made through a standard TCP/IP ethernet connection. The haptic device employed was a ground-based PHANToM manipulator. The server produced stable results with position gains over 1800 Newtons/meter on models containing as many as 24,000 polygonal primitives. The proxy update loop computation time was approximately $O(\log n)$, with n the number of polygons in the model.

8. STABILITY AND CONTROL

All the effects presented in the Section 6 were created solely by changing the proxy configuration. This reduces the job of the haptic controller to attempting to reduce the error between the proxy position and the haptic device. Position control of a mechanical system is a task that has been discussed extensively in the robotics literature. In our current implementation, we rely on a simple operational space proportional derivative (PD) controller.[19] As all the modeling effects are achieved by the movement of the proxy, controller gains and other parameters can be set solely by considering the mechanical properties of the haptic device. The stability properties of such controllers are well known and can be made quite robust if a sufficiently high update rate can be maintained.

A major advance of our approach is that the low-level control loop can be separated from the contact/proxy update loop to ensure that a high servo control rate can be maintained. As the position of the proxy depends on the models in the environment, which are supplied by the user application, it is not possible to guarantee real-time performance. If the control and update loops are separated, the haptic controller can be commanded at a fixed rate to use the last computed proxy position. Thus, the stability of the controller is maintained while the fidelity of the haptic display degrades gracefully as the complexity of the environment is increased.

It remains to show that the movement of the proxy is stable. As seen in Section 2, the basic move/update loop can only decrease the distance to the user's finger. It can therefore be shown that the update loop will add no energy to the user/haptic system. Likewise the static, dynamic, and viscous friction properties only restrict the motion of the proxy and are thus inherently stable. Shading and texture can increase the distance between the user's finger and the proxy. This increase implies that the surface is active and can add energy to the user/haptic system. In most graphic models, the interpolated and true surface normals typically differ by less than 30°. In these cases, the added energy is very small, and is not noticed by the user. In our test on typical models, the motion was always stable, although there do exist contrived examples in which unstable behavior is possible. Lastly, energy stored in a virtual dynamic system can be transferred to the user through contact. If the system being modeled is inherently stable then the entire system will be stable. Nevertheless the masses and inertias of the simulated system should be selected small enough that they cannot damage the haptic device or the user, and care should be taken to ensure that the motion of the simulated system is bounded.

9. APPLICATIONS

The intuitive nature of haptic interaction makes it well suited for a wide range of applications. For instance, haptics can be used to train a surgeon to perform an operation without the cost and difficulties of training on animals or cadavers. In another area, a haptic system can be used to allow an animator to specify the movement of a 3D model. The animator can feel the

Figure 12. Many types of haptics environments can be modeled by our system, from the simulation of a small microsensor to a large roller-coaster. Other models include a windmill, a crank, a rocket carousel, and the famous Cadwell teapot.

joint limits of the model and feel the penetration constraints imposed by the environment. In mechanical design, an engineer can apply force and interact with a model in a physically intuitive manner. Other applications are as yet unimagined, but it is hoped that their development will be spurred by the low-level work presented here.

Figure 12 illustrates some of the virtual environments that can be modeled by our system. On the upper left, a micromechanical sensor is modeled. The user is free to push on the test mass to see how the system responds to his/her input. The user can also use the probe to check clearances and ensure that the system will behave as expected. The size, mass, and time parameters of the system are scaled to allow intuitive interactions with the model. In other models in the figure, such as the windmill, roller-coaster, or carousel, the size and mass of the system is reduced so that the user can easily interact with an object that would be difficult to interact with in reality. Other objects like the crank and the teapot are rendered full size. These models illustrate the numerous possibilities for using haptics to interact with virtual systems.

10. CONCLUSION

The techniques we have described were used to model a variety of virtual models. As computational power continues to increase, the size and complexity of the models that can be simulated will continue to grow. By allowing a user to interact intuitively with a model, haptics can greatly improve efficiency in designing and evaluating new systems and designs. We are currently investigating methods to allow more complex systems to be modeled, and to allow interaction through more complex articulated effectors.

REFERENCES

1. R. Avila and S. Sobierajski, A haptic interaction method for volume visualization, Visualization '96 Proc, October 1996.
2. Y. Adachi, T. Kumano, and K. Ogino, Intermediate representation for stiff virtual objects, Proc IEEE Virtual Reality Annual Int Symp '95, March 1995, pp. 203–210.
3. D. Baraff, Analytical methods for dynamic simulation of non-penetrating rigid bodies, SIGGRAPH 89 Proc, August 1989, pp. 223–232.
4. D. Baraff, Fast contact force computation for nonpenetrating rigid bodies, SIGGRAPH 94 Proc, August 1994, pp. 23–34.
5. J. Blinn, Simulation of wrinkled surfaces, SIGGRAPH 89 Proc, August 1978, pp. 286–292.
6. P. Buttolo, D. Kung, and B. Hannaford, Manipulation in real virtual and remote environments, Proc IEEE Conf on Systems, Man and Cybernetics, August 1990, pp. 177–185.
7. K. Chang, Efficient dynamic control and simulation of robotic mechanisms, internal report.
8. J. Craig, Introduction to robotics mechanics and control, Addison-Wesley, 1989.
9. R. Featherstone, Robot Dyn Algorithms, Kluwer, 1987.
10. M. Finch, et al., Surface modification tools in a virtual environment interface to a scanning probe microscope, Proc 1995 Symp on Interactive 3D Graphics, April 1995, pp. 13–18.
11. H. Goldstein, Classical mechanics, Addison-Wesley, 1980.
12. E.G. Gilbert, D.W. Johnson, and S.S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space, IEEE J Robot Automat 4:(2) (1988).
13. P. Gill, S. Hannarling, W. Murray, M. Saunders, and M. Wright, User's guide to LLSOL, Stanford University Technical Report SOL86-1, January 1986.
14. S. Gottschalk, M.C. Lin, and D. Manocha, OBBTree: A hierarchical structure for rapid interference detection, SIGGRAPH 96 Proc, August 1996, pp. 171–180.
15. H. Gouraud, Continuous shading of curved surfaces, IEEE Trans Comput C-20:(6) (1971), 623–629.
16. H. Iwata and H. Noma, Volume haptization, IEEE 1993 Symp on Research Frontiers in Virtual Reality, October 1993, pp. 16–23.
17. T. Kane, Dynamics: Theory and applications, McGraw-Hill, 1985.
18. O. Khatib, Real-time obstacle avoidance for manipulators and mobile robot, Int J Robot Res 5:(1) (1986), 90–98.
19. O. Khatib, A unified approach to motion and force control of robotic manipulators: The operational space formulation, IEEE J Robot Automat 3:(1) (1987).
20. J.-C. Latombe, Robot motion planning, Kluwer, 1991, pp. 58–152.
21. M. Lin and J.F. Canny, A fast algorithm for incremental distance calculation, Int Conf on Robotics and Automation, May 1991, pp. 1008–1014.
22. W.R. Mark, S.C. Randolph, M. Finch, J.M. Van Verth, and R.M. Taylor, II, Adding force feedback to graphics systems: Issues and solutions, SIGGRAPH 96 Proc, August 1996, pp. 447–452.
23. T.M. Massie and J.K. Salisbury, The PHANToM haptic interface: A device for probing virtual objects. Dynamic Systems and Control 1994, Chicago, November 6–11, vol. 1, pp. 295–301.
24. M.D.R. Minsky, Computational haptics: The sandpaper system for synthesizing texture for a force-feedback display, Ph.D. dissertation, MIT, June 1995.
25. H.B. Morgenbesser, Force shading for haptic shape perception in haptic virtual environments. M. Eng. thesis, MIT, September 1995.
26. M. Ouh-Young, Force display in molecular docking, Ph.D. dissertation, University of North Carolina at Chapel Hill, UNC-CH CS TR90-004, Ferbuary 1990.
27. B.T. Phong, Illumination for computer generated pictures, Communications ACM 18:(6) (1975), 311–317.
28. S. Quinlan, Efficient distance computation between nonconvex objects, Int Conf on Robotics and Automation, April 1994.
29. D. Ruspini, K. Kolarov, and O. Khatib, Graphical and haptic manipulation of 3D objects, 1st PHANToM user's group workshop, September 27–30, 1996.
30. D. Ruspini, K. Kolarov, and O. Khatib, The haptic display of complex graphical environments, SIGGRAPH 97 Proc, August 1997, pp. 345–352.
31. D. Ruspini and O. Khatib, Collision/contact models for the dynamic simulation of complex environments, Workshop on Dynamic Simulation, IEEE/RSJ Int Conf on Intelligent Robots and Systems, IROS '97, Genoble, France, 1997.
32. S.E. Salcudean and T.D. Vlaar, On the emulation of stiff walls and static friction with a magnetically levitated input/output device, Dyn Syst Contr (1995), 123–130.
33. K. Salisbury, D. Brock, T. Massie, N. Swarup, and C. Zilles, Haptic rendering: Programming touch interaction with virtual objects, Proc 1995 Symp on Interactive 3D Graphics, April 1995, pp. 123–130.
34. M.A. Srinivasan, G.L. Beauregard, and D.L. Brock, The impact of visual information of the haptic perception of stiffness in virtual environments, ASME Winter Annual Meeting, November 1996.
35. R. Taylor, et al., The nanomanipulator: A virtual-reality interface for a scanning tunneling microscope, Proc SIGGRAPH 93, August 1993.
36. C. Zilles and J. Salisbury, Constraint-based god-object method for haptic display. Dyn Syst Contr 1 (1994), 146–150.