Haptic Rendering of Graphical Models

Diego Ruspini
Robotics Laboratory
Department of Computer Science,
Stanford University, Stanford, CA 94305-9010
`email: ruspini@cs.stanford.edu`

Abstract

In these notes we will examine the haptic rendering of the polygonal models commonly found in graphic applications. While much of the work described extends to other model representations, such as volumetric or NURB surfaces, these will be discussed in other sections of the course and will only be briefly mentioned here. The haptic rendering of models composed of large number of polygonal primitives is important because these models are the most widely used in creating interactive 3D environments. In this section we will look at how constraint-based methods can be used not only to enforce non-penetration constraints but also demonstrate how these methods can be applied to model properties such as shaded surfaces, friction and texture.

1 Introduction

A haptic interface is a force reflecting device which allows a user to touch, manipulate, create or alter objects in a simulated virtual world. Haptic rendering is the process by which a model specification is taken and appropriate forces are computed to give the illusion of physical contact through the haptic device. In these notes we will focus on haptic rendering of models that represent the surface of the environment using a large numbers of simple polygonal primitives. These bounding surface representations are important because they are the most common representations in use by interactive graphics systems. While many of the concepts presented in this section can be extended to work with other model representations (i.e. volumetric, implicit surfaces,...) these will be addressed in other sections and not explicitly examined here.

While many standards exist to specify the surface of a graphic model, at the lowest level, almost all graphics hardware is only capable of rendering simple polygons, lines and points. All higher order surfaces are decomposed into a set of polygonal patches before being rendered. The advantage of this approach is that the graphics hardware can be made highly optimized for displaying these simple primitives. In building a general purpose haptic rendering system it may be wise to follow the example of the graphics community and focus on a small set of powerful primitives then create a large base of low-level objects most of which will be never used in practice.

Towards this end, in these notes, we will examine some of the methods and algorithms that went into the development of the haptic rendering system "HL." The "HL" system shares many of the same ideas as other advanced haptic rendering systems and therefore forms a good basis for

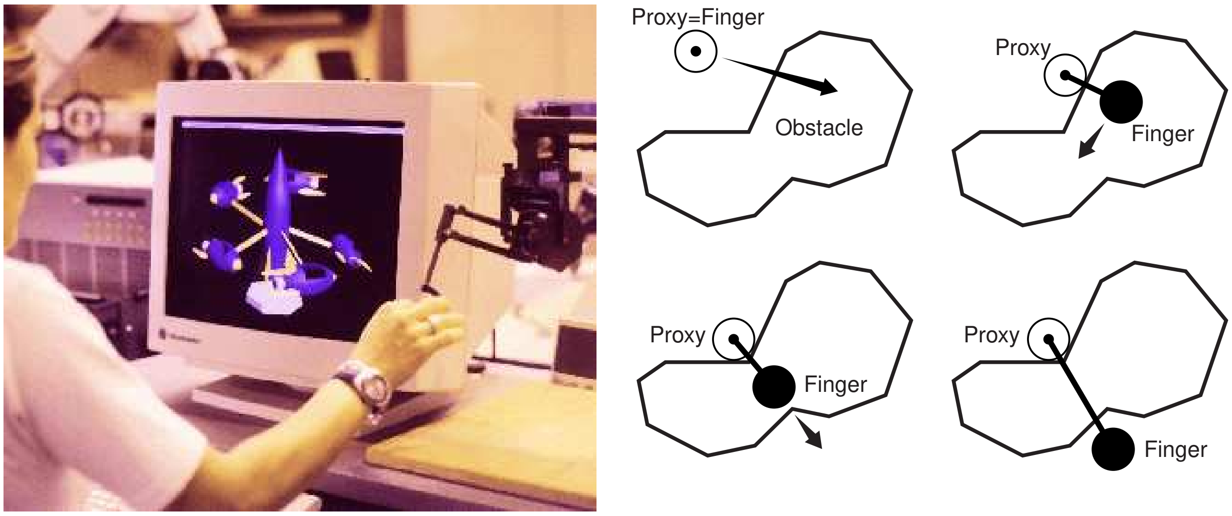

Figure 1: A user is haptically interacting with a dynamic virtual environment (left). The sensation of contact is created by applying forces through the haptic device to move the user's finger to the location of the constrained representative object *(the proxy)*. The virtual proxy moves to locally minimize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

examining how constraint-based haptic rendering can be accomplished. In addition to rendering polygonal geometry we will also look at how other graphic information can be used to augment the haptic sensation.

2 Penalty Based Methods

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.(Slide $\boxed{4}$)

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 then the one into which he or she originally penetrated. When this "pop-through" occurs the user will be actively pushed through the object, resulting in a 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.

3 Constraint Based Methods

Another approach first proposed by Zilles et. al [36] presents an alternative method that does not depend on determining the penetration distance into an obstacle. In constraint based methods a representative object substitutes in the virtual environment for the physical finger or probe. The representative object can be viewed as if connected to the user's real finger by a stiff spring. As the user moves his/her finger in the workspace of the haptic device he/she may pass into or through one or more of the virtual obstacles. The representative object, however, is stopped by the obstacles and quickly moves to a position that minimizes its distance to the user's finger position subject to the constraints in the environment. 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[5].

This representative object has been given many names (god-object, haptic point, ideal haptic interface point, surface contact point...) and has been used in a wide variety of systems. Each system differs in the way the environment is defined and how the update procedure is performed but most share some basic similarities. The update rate of the constrained object position must be very high ($> 1000Hz$) in order to achieve realistic high-fidelity force response. In each update loop the current position of the haptic device is found and the location of some representative effector point is computed. Some method of collision detection is utilized to determine if it is possible to move the representative object toward the effector configuration. The representative object is moved as far as possible by some simple motion and from there a new potential direction of motion is found. At the end of the servo loop the error between the user's position and the representative object is used to generate forces on the haptic device. This in effect reduces the error by physically moving the user's position to the configuration of the representative object.

In the system we will describe this representative object is designated as a *proxy*[6]. The virtual proxy was the first approach to look at rendering the "polygon soup" type models that are the most common in graphic applications. The virtual proxy is modeled as a finite massless sphere, and no topology of the objects in the environment is required making it applicable to dynamic environments and scene graphs containing many moving or intersecting obstacles. The virtual proxy framework was also extended to correctly and robustly utilize additional graphical information such as shading normals, friction and texture. An example of virtual proxy moving in the environment is illustrated in figure 1.

In the next section we will look at how the proxy's position is updated during each servo loop. In section 5 a method for simulated smooth curved surfaces using the information normally available for Phong or Gouraud shading is described. Methods to render static, dynamic and viscous friction are described in section 6. Texture is introduced in section 6.3. A short summary of how fast collision detection can be achieved and how dynamic models are introduced is described in sections 7 and 8. The conclusion shows some examples from the current system and describes some future directions.

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

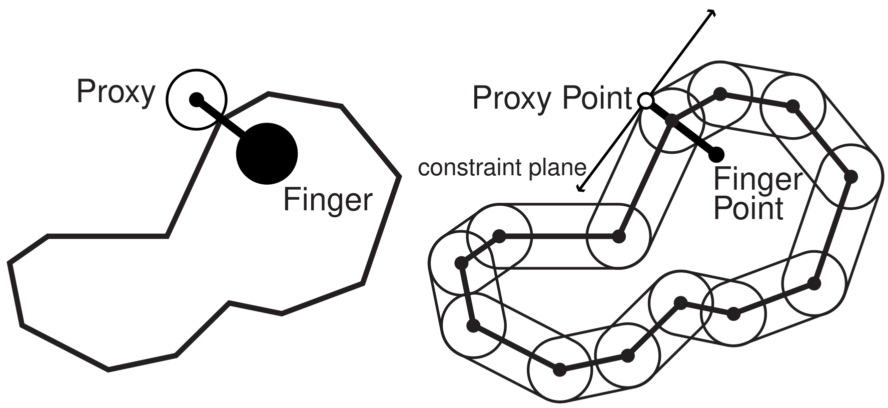

Figure 2: Actual and Configuration Space Obstacle

Many of the concepts used in haptic display have their origins in robotics applications. For this discussion we will model the proxy as a smooth massless sphere. The radius of the proxy is selected both 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, and checking and fixing a model to eliminate "leaks" is in general very computationally expensive. At each time step the position of the proxy is changed to reduce its distance to the user's finger position subject to the constraints in the environment. Each iteration is divided into two stages: *move* and *update*.

4.1 Moving the Proxy $\boxed{7}$

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 will change as dictated in the *update* stage. 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 and is described in detail in section 7. Many other methods of high-level collision detection have also been proposed [14]. All these techniques exploit spatial or temporal coherence to quickly eliminate primitives from the environment that lie too far from the proxy's path to effect 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 polyhedron. The low-level test is also similar to the ray intersection test used in ray-tracing applications for graphics. In our system we use Gilbert's algorithm since it requires very little preprocessing and can be used in other parts of the haptic rendering process as we will see below.

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.

4.2 Updating the Goal Position $\boxed{10}$

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 then be defined. Each constraint plane limits the potential motion of the proxy to the half-space above the plane. In addition the user's finger position will always be situated beneath the constraint plane. An example of the configuration space, C-obstacles, and the proxy constraint planes is shown in Figure fig:configurationspace.

The intersection of all such half-spaces defines a convex, unbounded polygon which 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 gilbert distance algorithm [12], used during collision detection (Section 7), can efficiently find the minimum distance between two convex bounded polyhedra. In addition to distance, the algorithm, will also find the nearest point on each body. Each polyhedra is defined as the convex hull of a set of points $P = p_1, \cdots, p_n$ and $Q = q_1, \cdots, q_m$. On completion the algorithm will return a set of weights w such that

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

Given that a polyhedra can contain only one point, finding the distance between a polyhedra and a single point is trivial.

As described above the problem can be stated as:

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

where x is the desired goal configuration, 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 all $\hat{n}_i$ are unit normals.

The specification of our free-space polyhedron, is not in a form that can be directly exploited by the Gilbert algorithm. The polyhedron is un-bounded and the limits of its extent are defined by the intersection of all the constraint planes, not the vertices of its convex hull. There does exist, however, a dual relationship between this free-space polyhedron and another polyhedron that does satisfy the requirements of the algorithm [11].

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 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. We will call the planes associated with these constraint surfaces the set of *active* planes. All other constraint planes will be considered *inactive*. At first we will assume 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 necessary to solve the problem.

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

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

where $N = \left[\hat{n}_{a_1} \ldots \hat{n}_{a_{m'}}\right], 0 \leq m' \leq 3$ is the matrix containing normals of the *active* constraint planes. Introducing the m' multipliers $\lambda = [\lambda_1 \cdots \lambda_{m'}]$, the Lagrangian for equation 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 equation 4 we obtain a dual for our original constraint equation 3:

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

Rewriting equation 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:

$$S' = \left[\begin{smallmatrix} 0 \\ 0 \\ 0 \end{smallmatrix} \middle| c_{a_1} \hat{n}_{a_1} \middle| \cdots \middle| c_{a_{m'}} \hat{n}_{a_{m'}} \right] \tag{9}$$

where $c_{a_i}, 0 \leq i \leq m'$ 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:

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

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

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

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

The solution to equation 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' algorithm. To prove that the solution lies in this space we must show that $\sum_{0 \leq i \leq m'} w_i = 1$ and $w_i \geq 0, \forall w_i, 0 \leq i \leq m'$ where w_i is the i^{th} term of vector w.

From the original problem statement $x_p = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ 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 $(u - x)^T (u - x) \leq \|u - x\| \leq \|u - x_p\| \leq 1$. Noting from equation 6 that $u - x = N\lambda$ and our original requirement that planes on the *active* set satisfy $n_i x = 0$, we see:

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

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 non-negativity 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

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

The constraint surface can only push, not pull, on the proxy. Therefore, the force normal to the surface must be non-negative ($n_i^T f_i = -k_s\lambda_i = -k_s c_i w_i \geq 0$). 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 easy 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 half-spaces 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 know 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 $\hat{u} = fracu\|u\|$.

Use Gilbert's algorithm to find the nearest point between $\hat{u}$ and a polyhedra defined by the $m+1$ of the dual space:

$$S = \left[\begin{smallmatrix} 0 \\ 0 \\ 0 \end{smallmatrix} \middle| c_1 \hat{n}_1 \middle| \cdots \middle| c_m \hat{n}_m \right] \tag{17}$$

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 non-zero 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 equation 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 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.

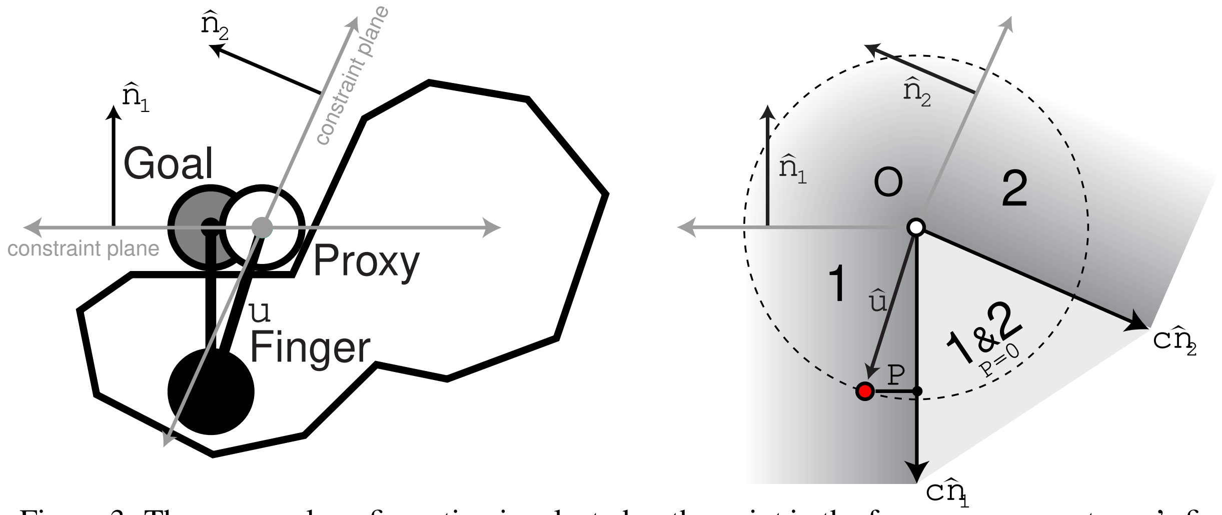

Figure 3: The new goal configuration is selected as the point in the free-space nearest user's finger position (left). Its equivalent dual space representation (right) which represents the change in potential caused by moving from the current configuration.

5 Haptic Shading $\boxed{12}$

As described above 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 models to allow regular polygonal surfaces to be perceived as if they were constructed out of curved continuous surfaces. As is illustrated in figure 4, in many graphic models, surface normals are defined at the vertices of a polygonal mesh which 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 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.

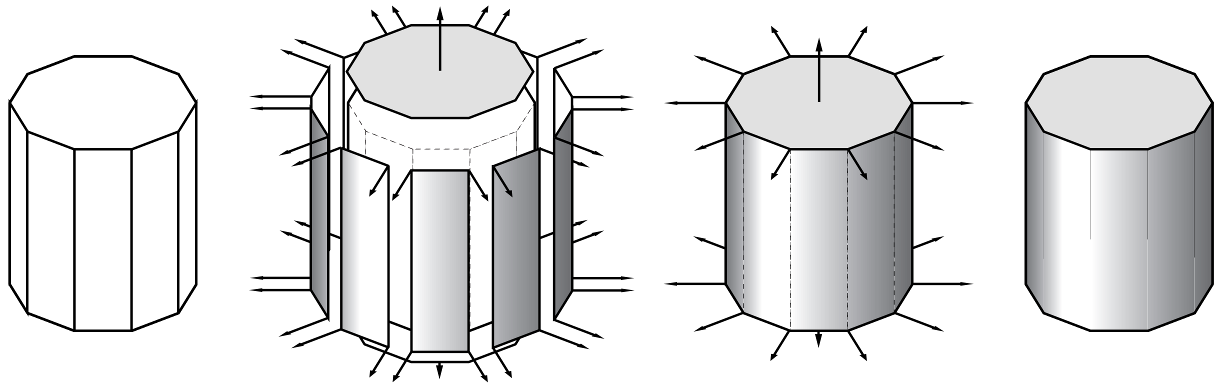

Figure 4: In most graphic systems curved objects are modeled as a set of flat polygonal patches (left). To achieve the appearance of a continuous surface, surface normals are defined on the vertices of each patch (center). The color or lighting model are interpolated over the surface to produce a continuously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

For haptics these given vertex surface normals can be used to give the sensation the user is touching a continuous, non-faceted surface. This modeling makes use of a haptic illusion first described by Minsky et al. [24]. Minsky 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 bases for the illusion derives from the disparity between a humans force and position differentiation capabilities. While humans are able to distinguish small force changes, they are relatively incapable 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.

Morgenbesser and Srinivasan [25] where 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 its applicability when the environment contained intersecting or moving obstacles. In addition, contact with multiple constraint surface 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. An alternative approach is to use an alternative minimization to determine the best goal position for the proxy. Since this 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 will be discussed in section 5.1. 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* stage described in section 4[13]. In this pass, however, the interpolated constraint plane is used instead of original for any contact surface containing user supplied normals. This new sub-goal 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 is called again but with the original (non-interpolated) constraint planes. The goal configuration generated in the first pass substitutes for the user's finger position[14]. This two pass approach has the effect of finding the nearest valid configuration to the minimal configuration as defined by the interpolated surface normals.

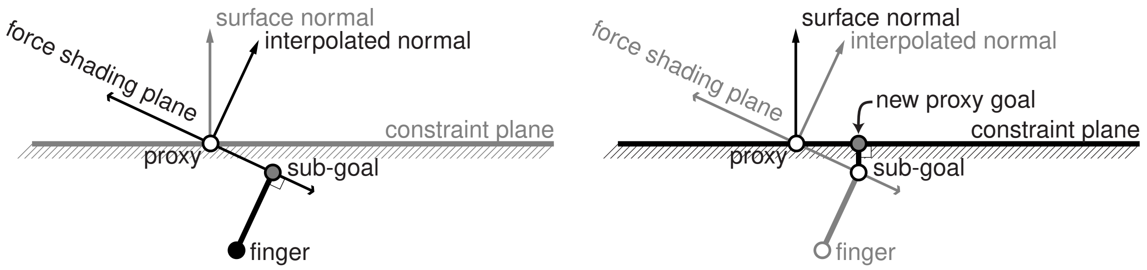

Figure 5: Two pass haptic shading with specified normals

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* stage no obstacle is encountered, the proxy will move to this configuration and a force pulling the user's finger to right will be applied as would be expected from a object having the surface normal illustrated.

If the sub-goal configuration, after the first pass, is above all the true constraint planes, the sub-goal is first projected back onto the nearest true constraint plane. This ensures that the new sub-goal 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 difference between the user's position and the position of the proxy are shown as the user's finger follows a circular counter-clockwise 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 which gives the user the impression of crossing over and 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 the true circular object illustrated in Figure 6(c) 15 . The affect of this minimization is to eliminate the large instantaneous changes in force that normally 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.

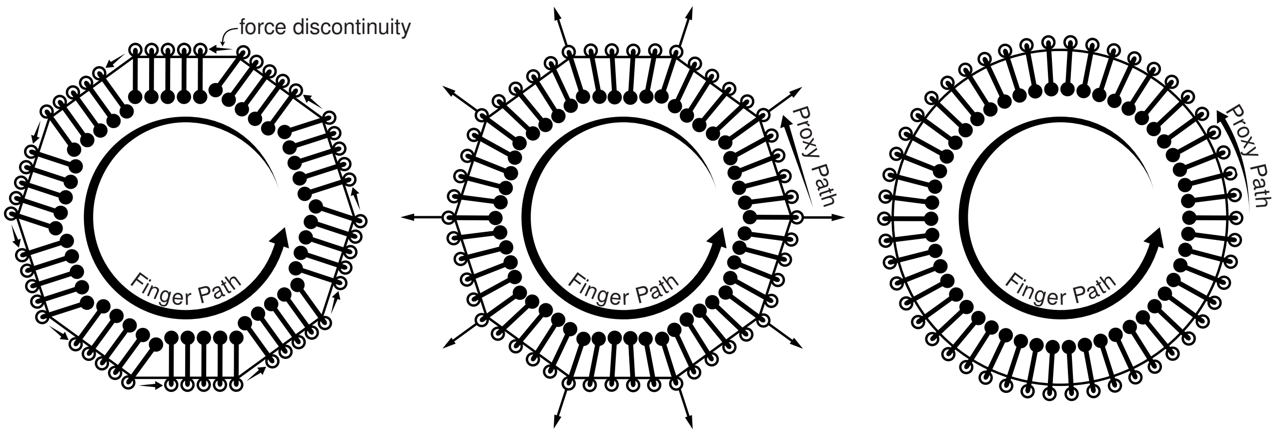

Figure 6: Haptic shading (center) eliminates the force discontinuities associated with moving along a faceted cylindrical surface (left). Although the path of the finger and proxy differ from that of a true cylinder (right) a humans position discrimination ability is insufficient to distinguish the tactile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

5.1 Determining Shaded Surface Normals

For the shading algorithm described above, 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 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 Gilbert et. al [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 the finding the interpolated normal for surface contact is fairly straight forward, special consideration needs to be given if the proxy is in contact with one of the edges or vertices of the polygon. As is illustrated in Figure 5.1 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 5.1(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 which 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 create 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.

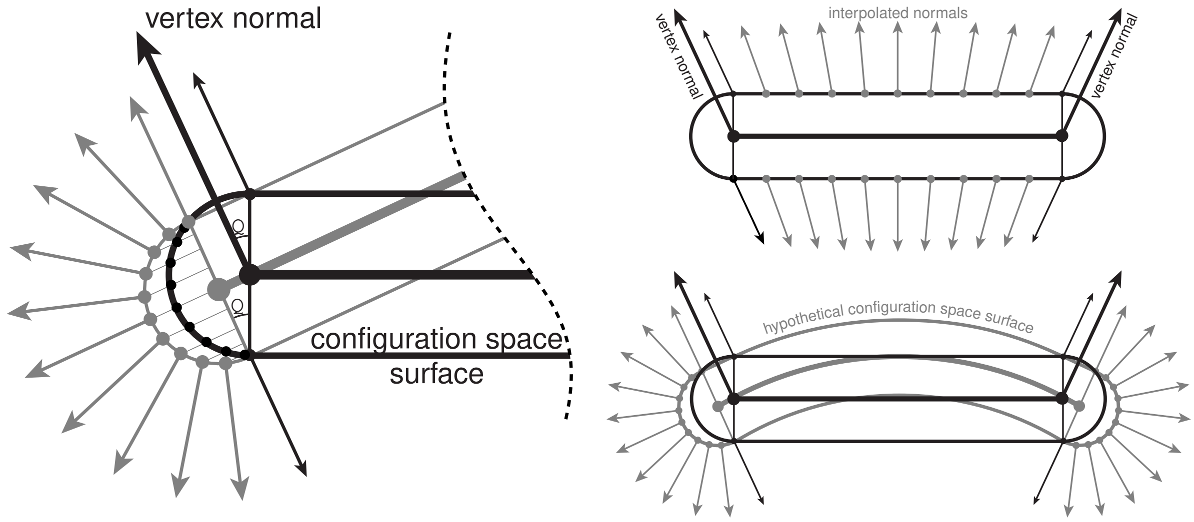

Figure 7: The surface normals of a hypothetical configuration space obstacle are projected onto the true configuration space surface to define the shading normals used to simulate a curved surface.

One approach that does not suffer the problems with the previously mentioned interpolation schemes is to attempt to model the surface normals that would be created if the surface was rotated. If contact is made with the boundary of a configuration space obstacle a hypothetical configuration space surface is created which 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 5.1(a). The configuration space normals of this hypothetical surface are projected onto the actual configuration space boundary too define the shaded surface normals for the configuration space obstacle. Some example configurations are shaded normals are illustrated in Figure 5.1(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 match at their configuration space boundary. This will ensure that the transition across the surface boundary will feel smooth to the user.

6 Surface Properties

Several researchers [2, 6, 22, 32, 33] have proposed methods to simulate static, dynamic, viscous friction and texture. These methods worked by introducing additional force to simulated the forces of friction by the contact surface and often depended on estimates of the finger's velocity which 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 controller that is much more stable and easier to control.

6.1 Static Friction $\boxed{16}$

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 $\|f_t\| \leq \|f_n\|$, i.e., the user's position is in the friction cone of the surface. An example of such a configuration is shown in figure 8(left). When any constraint surface is in static contact with the proxy, the proxy's position is prevented from changing by making the new sub-goal position equal to the current proxy position.

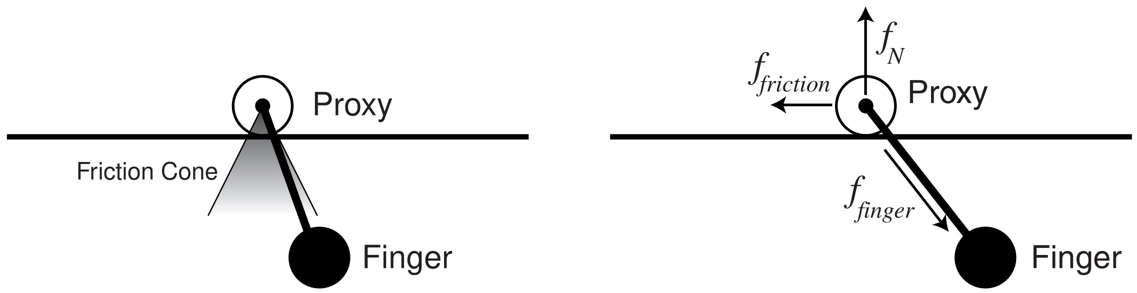

Figure 8: Static friction can be simulated by not permitting proxy movement if the user's finger is in a given friction cone (left). Viscous and dynamic friction can be modeled by constraining the motion of the proxy subject to the forces applied to it(right).

6.2 Viscous and Dynamic Friction $\boxed{17}\boxed{18}$

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

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

where x is the position of the proxy, m is its mass, b is the viscous damping term, and f_{finger},f_N, $-\mu f_N$ 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 equation 20 can be observed as $m \longrightarrow 0$. 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

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

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.

6.3 Texture $\boxed{19}$

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 then can be realistically created using polygonal surfaces alone. An image-based texture map can be used to modulate any of the surface properties described in section 6. In addition the force shaded constraint planes can be modified in a manner similar to bump mapping introduced by Blinn [5] for computer graphics. For this application 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 a displacements to the surface original or shaded normal as in Blinn. Once the texture normal is found it is used as the same as the shading normal in section 4. An example of this is 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.

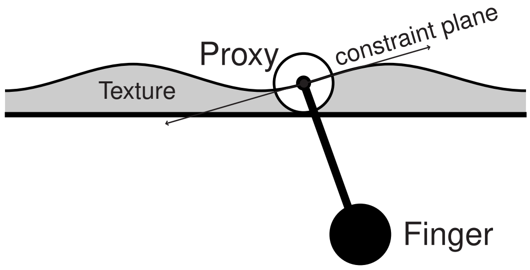

Figure 9: An image based texture is used to alter the constraint plane to create the sensation of bumps on the virtual surface

While imaged mapped texture is very similar to haptic shading some important differences exist. Texture is typically used to model higher frequency information then used in shading. In fact, in shading the primary goal is to eliminate the high frequency discontinuities at the surface boundaries. This necessitates several changes in-order to capture this high level detail and avoid missing important features that describe the surface. First, during the *move* stage of the proxy update loop the path of the proxy on the surface should be mapped onto the texture space and the values encountered on the texture surface should be checked to see if proxy's position should be constrained$\boxed{20}$. An example of this is illustrated in Figure 10(left). If the whole path is not examined a small detail may be missed.

Secondly, unlike bump-mapping for computer graphics it may be desirable to allow multiple constraint planes to be specified at a given location$\boxed{20}$. As is illustrated in Figure 10(right) where at a given point the proxy may be in contact with multiple surfaces. In this case it may be desirable to allow multiple texture images to be specified for a surface. One constraint plane is created for each image. In our system selection of appropriate textures images to model a given surface is currently left to the user.

These techniques are useful for modeling basic textures but much work still needs to be done.

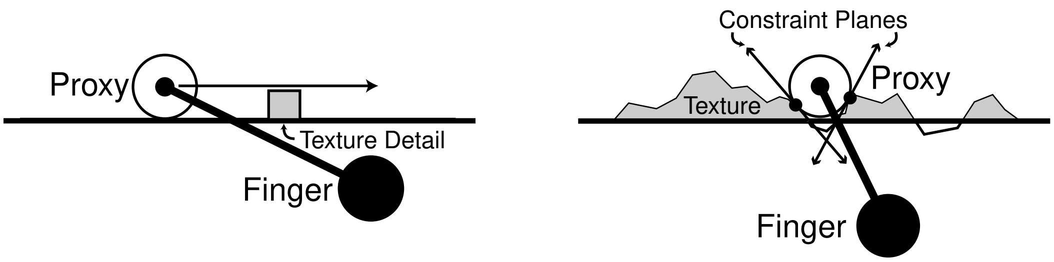

Figure 10: The entire path of the proxy must be checked to ensure small textural detail is not missed (left). Likewise multiple constraint planes may need to be defined to represent the constraints on the user's position as specified by the texture (right).

7 Collision Detection $\boxed{8}\boxed{9}$

Because the environment is normally constructed from a large number of primitives, a naive test based on checking if each primitive is in the path of the proxy would be prohibitively expensive. Instead a hierarchical bounding representation for the object can be constructed to take advantage of the spatial coherence inherent in the environment. The bounding representation, used by our system, is similar to that first proposed by Quinlan [28]. This approach represents a good example of how high-level pruning structure is constructed.

The 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 can be used to build the interior nodes of the tree. This algorithm works in a manner similar to quick-sort. 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 mid-point of the longest axes of the bounding box. Each of the resulting two subsets should be compact and contain approximately an 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 sub-trees as children. A cut-away view showing the leaf nodes (yellow) and bounding sphere hierarchy for a typical model is illustrated in Figure 11. Note that a node is not required to fully contain all the descendant internal nodes, only the descendant leaf nodes.

In Quinlan's approach, 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 mid-point of the bounding box already computed earlier. 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 $O(n\lg n)$ 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 $O(\lg n)$.

The sphere hierarchy is used to prune the number of low-level checks that need to 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

Figure 11: Bounding Sphere Hierarchy of a cat model

path of the proxy. 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 low-level distance algorithm can be kept in the cache to reduce the computation time between successive calls.

8 Dynamics $\boxed{21}$

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 is the vector representing the internal and external torques applied to the system either through internal actuation or external forces applied by the environment.

When a collision occurs, between the proxy and an object(s) 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

$$\Gamma_{ext} = J_i^T f_i, \tag{23}$$

where J_i is the Jacobian of the contact point i, such that the velocity v_i of the contact point is given by $v_i = J_i\dot{\Theta}$. The contact force f_i caused by contact of the proxy with the environment can be compute from the multipliers found in the *update* stage as defined by equation 19

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 [30]. 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.

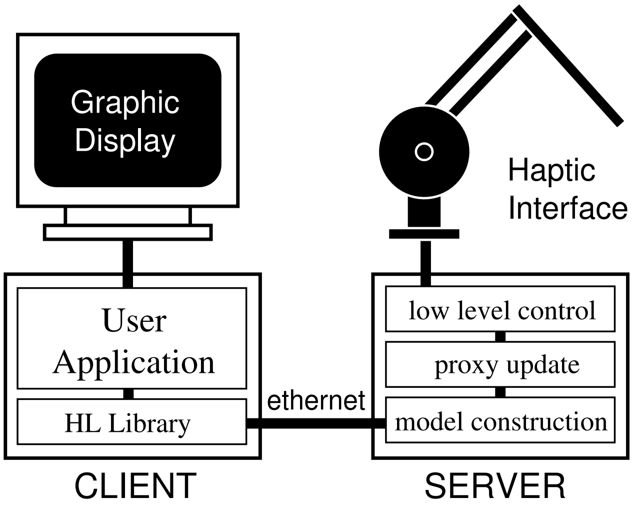

Figure 12: System architecture

9 Stability and Control 22

All the effect presented in the previous section 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. The 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 these types of controllers are well known and can be made quite robust if a sufficiently high update rate can be maintained.

Because the size of a model may be unknown at run-time, it is important to construct haptic render in such a way that haptic display degrades gracefully and safely as the complexity of the environment is increased. One approach is to separate the application, model construction, proxy update and the low level control tasks into separate processes. This will help insure that if there is a delay in one sub-system the problem will not effect overall performance and safety of the system. In the "HL" system illustrated in Figure fig:arch the scene graph is sent to the haptic server on a separate machine by the application program. If a failure occurs in the application program the current scene graph can continue to be used until a timeout occurs and the haptic rendering process can be shutdown gracefully. The haptic server can itself be divided into separate tasks to increase the level of robustness and safety.

The computation of the bounding sphere hierarchy (BSH), while fast, is not real-time since it is a function of the number of primitives in the object. As the transmission and computation of a BSH for an object cannot be guaranteed to be completed in one servo tick it is important to keep the previous definition of the object structure until such time as the new definition is available. The robustness of the system is increased because while the addition of an object in a scene might be delay it will never be only partially defined.

As is the case with the construction of the BSH the update of the proxy position is also a function of the complexity of the environment. In this case, thankfully, $O(lgn)$ instead of $O(nlgn)$. As such it still can not be made real-time. There does exist, however, a point which is in the free space of the obstacles which can be used when no new position is available. This point is the last computed position of the proxy. 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 the movement of the proxy is stable. As seen in section 4 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 then $30°$. 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 where unstable behavior is possible. Lastly, energy stored in a virtual dynamic system can be transfered 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 so as not to be so large that they may damage the haptic device or the user, and care should be taken to ensure the motion of simulated system is bounded.

10 Applications 23

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 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 can not even yet be imagined but it is hoped their development will be spurred by the low-level work presented here.

Figure 13 illustrates some of the virtual environments that can be modeled by our system. On the upper left a micro-mechanical sensor is modeled. The user if 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 Figure 13 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 which would be difficult to interact with in reality. Other object like the crank and the teapot are rendered full size. These models illustrate the numerous possibilities for using haptics to interact with virtual systems.

In tests and the examples below the client computer was a SGI Indigo2 running IRIX 6.2. The server was a 200MHz 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 24000 polygonal primitives. The proxy update loop computation time was approximately $O(\lg n)$ with the number n of polygons in the model.

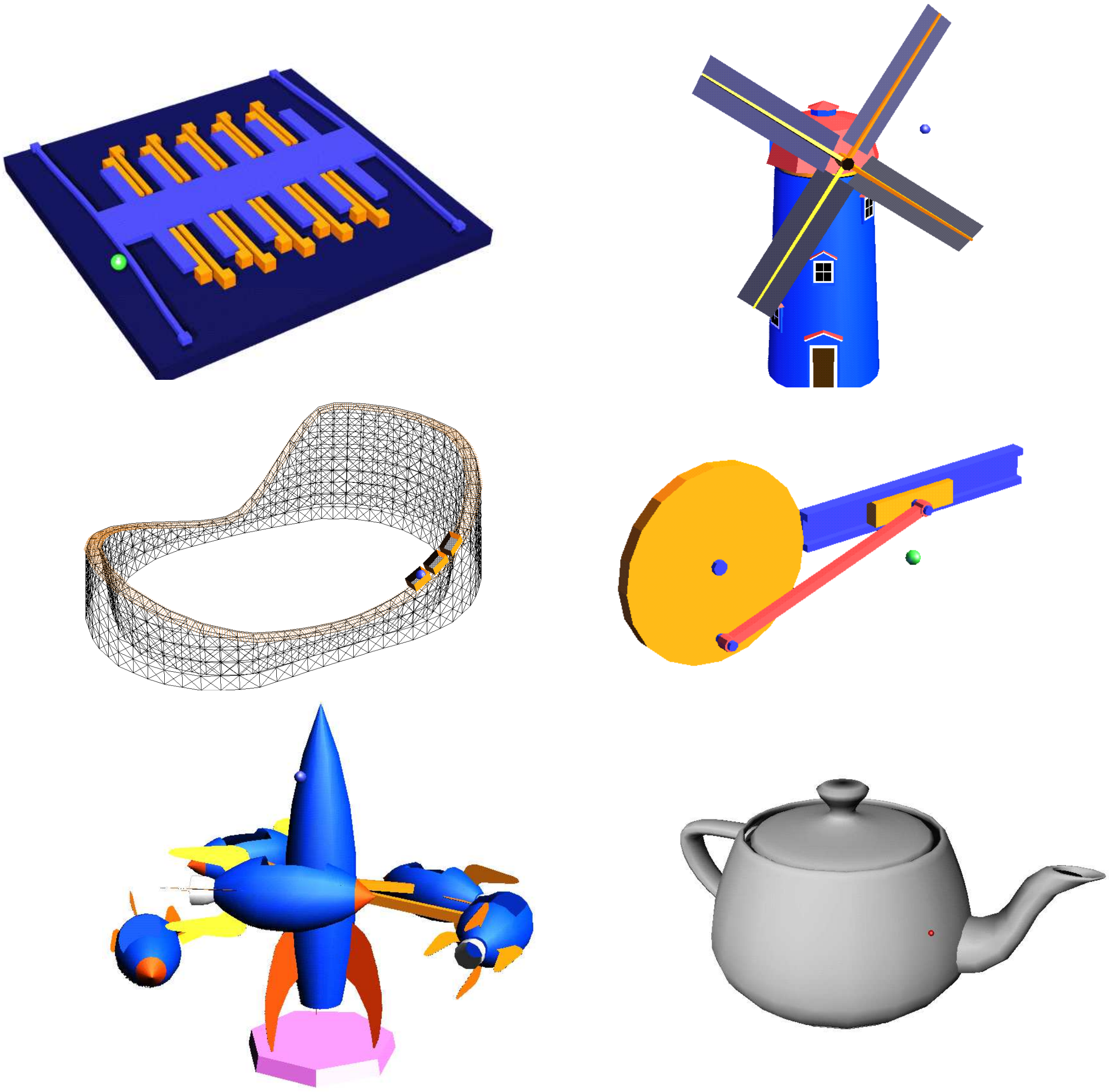

Figure 13: Many types of haptics environments can be modeled by our system from the simulation of a small micro-sensor to a large roller-coaster. Other models include a windmill, a crank, a rocket carousel or the famous Cadwell teapot.

11 Conclusion

The techniques we have described were used to model a variety of virtual models, see Figure 13. As computational power continues to increase the size and complexity of the models that can be simulated will continue to grow. Haptics by allowing a user to interact intuitively with a model can greatly improve the efficiency in designing and evaluating new systems and designs. We are currently investigating methods to allow more complex systems to be modeled and allow interaction through more complex articulated effectors.

References

[1] Avila, R., Sobierajski, S., "A Haptic Interaction Method for Volume Visualization," *Visualization '96 Proceedings*, October 1996.

[2] Adachi, Y., Kumano, T., Ogino K., "Intermediate Representation for Stiff Virtual Objects." *Proc. IEEE Virtual Reality Annual International Symposium '95*, (March 1995), pp. 203-210.

[3] Baraff, D., "Analytical Methods for Dynamic Simulation of non-penetrating Rigid Bodies," *SIGGRAPH 89 Proceedings*, (August 1989), pp. 223-232.

[4] Baraff, D., "Fast Contact Force Computation for Nonpenetrating Rigid Bodies," *SIGGRAPH 94 Proceedings*, (August 1994), pp. 23-34.

[5] Blinn, J., "Simulation of Wrikled Surfaces," *SIGGRAPH 89 Proceedings*, (August 1978), pp. 286-292.

[6] Buttolo, P., Kung, D., Hannaford, B., "Manipulation in Real Virtual and Remote Environments," *Proc. IEEE Conference on Systsms, Man and Cybernetics*, (August 1990), pp. 177-185.

[7] K. Chang, "Efficient Dynamic Control and Simulation of Robotic Mechanisms," internal report.

[8] Craig, J., "Introduction to Robotics Mechanics and Control," *Addison-Wesley*, 1989.

[9] Featherstone, R., *Robot Dynamics Algorithms.* Kluwer, 1987.

[10] Finch, M., Chi, V., Taylor, R. M. II, Falvo, M., Washburn, S., Superfine, R., "Surface Modification Tools in a Virtual Environment Interface to a Scanning Probe Microscope," *Proc. 1995 Symposium on Interactive 3D Graphics*, (April 1995), pp. 13-18.

[11] H. Goldstein, "Classical Mechanics," Addison-Wesley Publishing Company Inc., 1980.

[12] Gilbert, E. G., Johnson, D. W., Keerthi, S. S., "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," *IEEE Journal of Robotics and Automation*, Vol. 4, No. 2, April 1988.

[13] Gill, P., Hannarling, S., Murray, W., Saunders, M., Wright, M., "User's Guide to LLSOL," *Stanford University Technical Report SOL86-1*, January 1986.

[14] Gottschalk, S., Lin, M. C., Manocha D., "OBBTree: A Hierarchical Structure for Rapid Interference Detection," *SIGGRAPH 96 Proceedings*, (August 1996), pp. 171-180.

[15] Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Transactions on Computers*, C-20(6):pp 623-629, June 1971.

[16] Iwata. H., Noma, H., "Volume Haptization," *IEEE 1993 Symposium on Research Frontiers in Virtual Reality*, pp. 16-23, October 1993.

[17] T. Kane, *Dynamics: Theory and Applications*, McGraw-Hill, 1985.

[18] Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robot," *International Journal of Robotics Research*, 5(1), pp. 90-98.

[19] Khatib, O., "A Unified Approach to Motion and Force Control of Robotic Manipulators: The Operational Space Formulation," *IEEE Journal of Robotics and Automation*, Vol 3., No 1., 1987.

[20] Latombe, Jean-Claude, "Robot Motion Planning," Kluwer Academic Publishing, 1991, pp. 58-152.

[21] Lin, M., Canny, J. F., "A Fast Algorithm for Incremental Distance Calculation," *International Conference on Robotics and Automation*, (May 1991), pp. 1008-1014.

[22] Mark, W. R., Randolph, S. C., Finch, M., Van Verth, J. M., Taylor, R. M. II, "Adding Force Feedback to Graphics Systems: Issues and Solutions," *SIGGRAPH 96 Proceedings*, (August 1996), pp. 447-452.

[23] Massie, T. M., Salisbury, J. K., "The PHANToM Haptic Interface: A Device for Probing Virtual Objects." ASME Haptics Interfaces for Virtual Environment and Teleoperator Systems, *in Dynamic Systems and Control 1994*, (Chicago, Nov. 6-11), vol. 1, pp. 295-301.

[24] Minsky, M. D. R., "Computational Haptics: The Sandpaper System for Synthesizing Texture for a Force-Feedback Display," PhD thesis, MIT, June 1995.

[25] Morgenbesser, H. B., "Force Shading for Haptic Shape Perception in Haptic Virtual Environments." M.Eng. thesis, MIT, September 1995.

[26] Ouh-Young, M., "Force Display in Molecular Docking," *Ph.D. Dissertation*, University of North Carolina at Chapel Hill, UNC-CH CS TR90-004, February, 1990.

[27] Phong, B. T., "Illumination for Computer Generated Pictures." *Communications of the ACM,* 18(6),(June 1975), pp. 311-317.

[28] Quinlan, S., "Efficient Distance Computation between Non-Convex Objects," *Int. Conference on Robotics and Automation,* April 1994.

[29] Ruspini, D., Kolarov, K., Khatib, O., "The Haptic Display of Complex Graphical Environments." SIGGRAPH 97 Proceedings, (August 1997), pp. 345-352.

[30] Ruspini, D., Khatib, O., "Collision/Contact Models for the Dynamic Simulation of Complex Environments," *Workshop on the Dynamic Simulation, IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, IROS '97, Genoble, France, 1997.

[31] Ruspini, D., Kolarov, K., Khatib, O., "Graphical and Haptic Manipulation of 3D Objects." *First PHANToM User's Group Workshop*, September 27-30, 1996.

[32] Salcudean, S. E., Vlaar, T. D., "On the Emulation of Stiff Walls and Static Friction with a Magnetically Levitated Input/Output Device," ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems, in *Dynamics Systems and Control*, (April 1995), pp. 123-130.

[33] Salisbury, K., Brock, D., Massie, T., Swarup, N., Zilles, C., "Haptic Rendering: Programming Touch Interaction with Virtual Objects," *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, (April 1995), pp. 123-130.

[34] Srinivasan, M. A., Beauregard, G. L., Brock, D. L., "The Impact of Visual Information of the Haptic Perception of Stiffness in Virtual Environments," *ASME Winter Annual Meeting*, November 1996.

[35] Taylor, R., Robinett, W., Chi, V., Brooks, F., Wright, W., Williams, R., Snyder, E., "The Nanomanipulator: A Virtual-Reality Interface for a Scanning Tunneling Microscope," *Proceedings of SIGGRAPH 93*, August 1993.

[36] C. Zilles, J. Salisbury, "Constraint-based God-object Method for Haptic Display." ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1994, *Dynamic Systems and Control 1994*, vol. 1, pp. 146-150.