Six-Degree-of-Freedom Haptic Rendering Using Incremental and Localized Computations

Young J. Kim Miguel A. Otaduy Ming C. Lin Dinesh Manocha

Department of Computer Science
University of North Carolina at Chapel Hill
{youngkim,otaduy,lin,dm}@cs.unc.edu

Abstract

We present a novel six-degree-of-freedom haptic rendering algorithm using incremental and localized contact computations. It uses an incremental approach for contact and force computations and takes advantage of spatial and temporal coherence between successive frames. As part of a preprocess, we decompose the surface of each polyhedron into convex pieces and construct bounding volume hierarchies to perform fast proximity queries. Once the objects have intersected, we compute the penetration depth (PD) in the neighborhood of the contact between each pair of decomposed convex pieces using a new incremental algorithm. Moreover, we cluster different contacts based on their spatial proximity to speed up the force computation. We have implemented this algorithm and applied it to complex contact scenarios consisting of multiple contacts. We demonstrate its effectiveness on electronic prototyping of complex mechanical structures and virtual exploration of a digestive system.

Keywords: 6-DOF haptic rendering, penetration depth, proximity query, clustering

1 Introduction

Virtual environments and intelligent systems require intuitive interfaces for man-machine interaction. These may include visual, auditory, and haptic interfaces. However, compared to the presentation of visual and auditory information, techniques for haptic display have not been as well developed. Haptic display is often rendered through what is essentially a small robot arm, used

in reverse. Such devices are now commercially available for a variety of configurations (2D, 3D, specialized for laparoscopy or general-purpose) [Burdea 1996].

Some of the commercially available and commonly used haptic devices include the 3-degree-of-freedom (3-DOF) PHAMToM arm [Massie and Salisbury 1994], SARCOS Dexterous Arm [Nahvi et al. 1998], etc. Using these devices for interacting with virtual objects involves computing point-object contacts. They typically provide only force feedback and do not offer sufficient dexterity and control for applications like virtual prototyping (e.g. assembly planning and maintainability studies), medical simulation and teleoperation, which need to simulate arbitrary object-object interactions. A 6-DOF force-feedback device, such as the 6-DOF PHANTOM^{TM} arm, can provide torque feedback in addition to force display within a large translational and rotational range of motion. However, the application of these high-DOF devices has been limited. This is mainly due to the complexity of accurate calculation of all contacts and restoring forces that must be computed at the desired force updates (typically a few hundred Hz or higher). Current haptic rendering algorithms developed for 3-DOF haptic devices primarily deal with single point contact with the virtual models [Gregory et al. 1999; Colgate et al. 1995; Ruspini et al. 1997; SensAble Technologies 1997; Thompson et al. 1997] and are not directly applicable for computing object-object contacts.

There is a vast literature on rigid and deformable body dynamics for computing forces upon impact. However, the real-time performance constraint of haptic rendering limits the complexity of the algorithms that can be used. In practice, penalty methods are often chosen to compute contact forces due to their simplicity and low computational cost. When using a penalty based method, we need to first define a penetration potential energy that measures the amount of intersection between two models. One of the accurate measurements of the amount of intersection is the penetration depth, commonly defined as the minimum (translational) distance required to separate two intersecting (rigid) objects. However, no general and efficient algorithms are known for computing penetration depth between polyhedral objects for haptic rendering. The well-known algorithms based on computing the Minkowski sum of the polyhedra can have $O(n^6)$ worst case complexity, where n is the number of features [Dobkin et al. 1993]. Some of the proposed solutions for 6-DOF haptic rendering use either voxel sampling [McNeely et al. 1999] or prediction methods

[Gregory et al. 2000] to compute the response force, which can result in force discontinuities or unstable behavior. Pre-contact forces are also used along with penalty forces, to reduce the amount of penetration and increase the stability of the computation [McNeely et al. 1999]. Recently, a new algorithm has been proposed for global penetration estimation between polyhedral models [Kim et al. 2002b]. However, this approach can take a few seconds to estimate PD and is not fast enough for haptic display.

1.1 Main Results

In this paper we present a novel 6-DOF haptic display algorithm using localized contact computations. It includes:

- A localized force model formulated by clustering nearby surface contacts. It reduces the overall computational cost and improves the stability of force updates. This force model is based on penalty methods and penetration depth (PD) estimation, and can easily include force shading and various friction models.

- A hierarchical proximity query algorithm that quickly localizes the contact regions and computes the disjoint and intersection measures. It decomposes each non-convex polyhedron into a collection of convex pieces and constructs a bounding volume hierarchy for collision detection and contact analysis.

- A novel, fast incremental method for estimating penetration depth (PD) between convex polyhedra using an iterative local optimization method. The algorithm finds a "locally optimal solution" by walking on the surface of the Minkowski sum, implicitly computed by constructing a local Gauss map between two polytopes.

- An estimation algorithm that progressively refines the PD values for general polyhedra based on PD computation between convex polytopes. It computes localized PDs between overlapping convex polytopes.

These approaches take advantage of motion coherence due to high force update rates and spatial locality around the neighborhood of the contact regions. In practice, they result in significant speed ups in contact force computation and PD estimation between general polytopes. We have tested our 6-DOF haptic rendering algorithm on several benchmarks and complex contact scenarios. We demonstrate its application to virtual exploration of a digestive system and rapid prototyping of complex mechanical structures. Our algorithm has been able to sustain the desired force update rates on moderately complex scenarios with *multiple* contacts.

1.2 Organization

The rest of the paper is organized in the following manner. We briefly survey the state of the art in Section 2. In Section 3, we give an overview of our haptic display algorithm. Section 4 presents our force computation model and we describe a novel penetration depth algorithm in Section 5. Section 6 discusses the system implementation issues and demonstrates the effectiveness of our algorithm on virtual exploration and rapid prototyping.

2 Previous Work

In this section, we briefly review previous work related to force computation, contact queries, distance computation and penetration depth estimation.

2.1 Contact Force Computation

Different techniques have been proposed for rigid body dynamics and contact force computation. Usually the choice of method depends on the application. Constraint-based dynamics computes contact response based on formulating it as a linear complementarity problem (LCP) [Baraff 1994]. Although this approach can compute more accurate responses, it is not suitable for complex haptic scenarios. LCP methods are based on finding exact contacts between the rigid bodies. Collisions and resting contacts are handled differently. When the relative velocity at the contact is higher than a given threshold, the contact is considered as a collision, and an impulse is applied to the colliding

objects. This produces an instantaneous change in the velocity. On the other hand, contacts with low relative velocity are solved according to the LCP approach. Finding exact collisions might not be feasible in real-time applications, and a large number of simultaneous collisions can be rather difficult to handle.

Some of the commonly used algorithms for fast computation of contact forces are based on penalty-based methods [McKenna and Zeltzer 1990; Moore and Wilhelms 1988]. Penalty forces are usually computed as elastic forces which depend on the inter-penetration between the objects. The main problem with penalty methods is that they can cause instabilities or unwanted vibration. This can happen if the stiffness of the contacts is too high, or if the force update rate is not high enough. Pre-contact braking forces [McNeely et al. 1999], which have a viscous nature, can also be considered as an example in this category. In practice, performing separation distance queries between disjoint objects is less expensive as compared to computing the penetration depth. As a result, pre-contact forces reduce the overall computational cost by reducing the amount of inter-penetration between the objects.

In the typical 3-DOF haptic display, the simulation is based on the interaction of a point (i.e. the tip of the probe) with the virtual objects in the scene [Ho et al. 1999]. Forces can be computed using constraint-based techniques such as the god-object [Zilles and Salisbury 1995] and virtual proxy [Ruspini et al. 1997], as well as intermediate representations like the intermediate plane [Adachi et al. 1995; Mark et al. 1996]. Other techniques, including ray-based rendering [Basdogan et al. 1997], have also been proposed.

For 6-DOF haptic rendering, both volumetric approaches [McNeely et al. 1999] and prediction methods for polygonal models [Gregory et al. 2000] have been proposed. The forces acting on the haptic probe can also be directly displayed to the user, or through an intermediate representation such as the virtual coupling [Adams and Hannaford 1998]. Our algorithm uses the same collision detection algorithm as described in [Gregory et al. 2000]. However, the haptic rendering algorithm presented in [Gregory et al. 2000] does not compute a good estimate of penetration depth for its penalty-based contact force model and instability can often arise within the non-convex region.

2.2 Collision and Distance Computations

The problems of collision detection and distance computations are well studied in computational geometry, robotics, simulated environments and haptics. Most of the prior work can be categorized based on the types of models: convex polytopes and general polygonal models.

For convex polytopes, various techniques have been developed based on linear programming [Seidel 1990], incremental computation of Minkowski sum [Cameron 1997; Gilbert et al. 1988], feature tracking based on Voronoi regions [Lin and Canny 1991; Mirtich 1998] and multi-resolution methods [Ehmann and Lin 2000; Guibas et al. 1999]. Some of these algorithms are based on incremental computations and exploit frame-to-frame coherence [Cameron 1997; Lin and Canny 1991; Mirtich 1998].

For general polygonal models, bounding volume hierarchies (BVH's) have been widely used for collision detection and separation distance queries. Different hierarchies differ based on the underlying bounding volume or traversal schemes. These include the AABB trees [Beckmann et al. 1990], OBB trees [Gottschalk et al. 1996], sphere trees [Hubbard 1995], k-DOPs [Klosowski et al. 1998], Swept Sphere Volumes [Larsen et al. 1999], and convex hull-based trees [Ehmann and Lin 2001].

2.3 Penetration Depth Computation

A few efficient algorithms to compute the penetration depth (PD) between convex polytopes have been proposed. Dobkin et al. computed the directional PD using Dobkin and Kirkpatrick polyhedral hierarchy [Dobkin et al. 1993]. For any direction d, it finds the directional PD for two convex polyhedra with n and m vertices in $O(\log n \log m)$ time. Agarwal et al. used a randomized approach to compute the PD [Agarwal et al. 2000]. Its running time is bounded by $O(m^{\frac{3}{4}+\epsilon}n^{\frac{3}{4}+\epsilon}+m^{1+\epsilon}+n^{1+\epsilon})$ for any positive constant ϵ. However, we are not aware of any implementations of these algorithms and not much is known about their behavior in practice.

Given the complexity of PD computation, a number of approximation approaches have been proposed to estimate them efficiently. Cameron had proposed an extension to GJK algorithm to compute upper and lower bounds for PD for convex polytopes [Cameron 1997]. This has been further

extended by Bergen, who improves the PD estimate by expanding a polyhedral approximation of the Minkowski sum of two polytopes [Bergen 2001].

Other approximation approaches are based on discretized distance fields. These include an approach based on graphics rasterization hardware and multi-pass rendering approaches [Hoff et al. 2001] as well as the use of fast marching level-set algorithms [Fisher and Lin 2001].

3 Overview of the Haptic Rendering Algorithm

In this section, we give an overview of our six-degree-of-freedom haptic rendering algorithm.

In our haptic rendering pipeline, we compute the displayed force based on the following steps:

1. Identify the convex pieces of the objects that are inter-penetrating or are closer than a distance tolerance.

2. **Penetrating Contact**: For each overlapping pair of convex pieces, compute its PD along with the associated PD features[1].

3. **Near Contact**: For each pair of convex pieces that are disjoint but within a given tolerance, declare a near contact and return the corresponding pair of closest features and their separation distance.

4. Cluster all contacts based on their proximity and their PD or distance values.

5. Compute penalty-based restoring forces at the clustered contacts.

Figure 1 shows the pipeline of our haptic display algorithm.

As a pre-processing step, we decompose each non-convex polyhedral object into a collection of convex pieces. We use a convex surface decomposition based on a variant of breadth-first-search. Convex pieces are then formed by taking the convex hull of each surface patch. We compute a bounding volume hierarchy (BVH) for each object using the convex hull as the underlying bounding volume. Each convex piece computed using the convex surface decomposition becomes a leaf node

[1]By the PD features, we mean a pair of features on both objects whose supporting planes realize the PD.

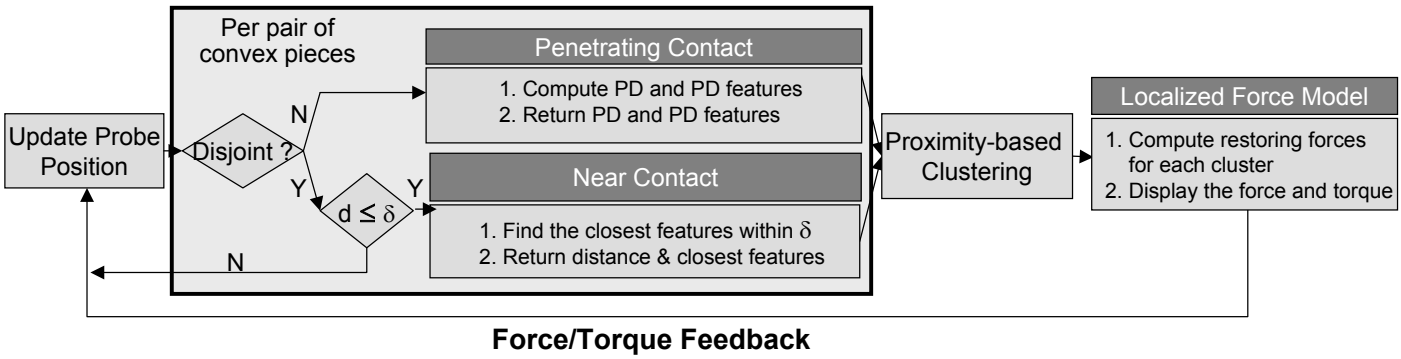


Figure 1: Haptic Display Pipeline

in the BVH. We recursively compute the internal nodes in a bottom-up manner. For more detailed discussion on the hierarchy and convex surface decomposition, please refer to [Ehmann and Lin 2001]. Using BVH's, both separation and intersection of objects are handled uniformly.

Our 6-DOF haptic rendering algorithm first checks whether convex pieces of two objects are overlapping or disjoint but within a given tolerance threshold value. At run time, the intersection test is recursively applied to nodes in one BVH against nodes in another BVH. We use an efficient, incremental algorithm for convex polytopes [Lin and Canny 1991] based on *Voronoi Marching* to perform a collision query on each pair of convex pieces. It computes the separation distance between the given pair. This top-down traversal is applied recursively to both the hierarchies until there is no intersection between the leaf nodes or until there is no leaf node within the tolerance value.

If two convex pieces are disjoint and inside the tolerance, we determine the closest features between them along with their associated distance measures (Near Contact). The features may correspond to a vertex, an edge or a face. If the pieces are overlapping, we identify the intersection regions and estimate the penetration depth (PD) and the associated PD features (Penetrating Contact). Each pair of closest features or PD features correspond to a single contact. The forces applied at near contacts can be regarded as elastic pre-contact forces. They reduce the amount of inter-penetration between objects, therefore increasing the robustness of our PD estimation.

Once we identify all the contacts, we cluster them based on some Euclidean distance δ between them. We use octrees to efficiently cluster the contacts. Then, for each clustered contact, we

compute its position, distance value and direction for the force in terms of a weighted average, where the weights are the distance values associated with each pair of contacts in the cluster. Finally, for each representative (clustered) contact, force and torque are independently computed, and the net force and torque are applied to the haptic probe by summing up the forces and torques of all the representative contacts.

4 Force Computation Model

In this section we describe how the restoring forces are computed, given the localized PD's and the corresponding features. We first define "contacts" between two surfaces, in the context of our force computation model. Since the performance of force computation depends on the number of pairwise contacts, we present an approach to improve the running time and the stability of force computation by clustering the contacts. Finally, we describe our force computation model and additional techniques to provide higher quality haptic feedback.

4.1 Contact Formulation

Exact contact between two non-convex surfaces is described by a set of 1D or 2D geometric features lying on the boundary of each polyhedra. However, computation of exact contact can be expensive for even simple contact scenarios. For disjoint cases when two objects are nearly in contact but separated, we can formulate the contact regions as the volume within a pre-defined distance tolerance. On the other hand, for penetrating contacts, we consider the actual intersection or penetration volume. In both cases, all features that lie within these volumes can be used to compute an explicit representation of the contact region. In practice, such an approach for exact computation of the contact regions can be relatively slow for haptic applications.

Instead, we sample the contact regions, and compute forces at the sampled contacts, which are added to yield the global forces to be applied to the objects. We define a "contact" between two objects as a pair of points, one from each object, that are within a user-defined tolerance. We use a convex surface decomposition algorithm to decompose each polyhedra into convex pieces and

compute the contact points between each pair of overlapping convex pieces.

When the objects are not overlapping, the algorithm computes the closest points on the boundary of each polyhedra using a hierarchical algorithm [Ehmann and Lin 2001]. In this case, the *contact points* corresponding to a pair of *closest points*. When two objects are inter-penetrating, the contact points are defined by the features used to define the penetration depth and the separating planes that support those features (described in more detail in Section 5). The resulting set of contact points represents a superset of the local minima of the distance function between the objects around the contact area. In practice, they turn out to be a good approximation for force computation.

For every contact c_i, the contact determination module returns: the contact point on Object1, $\boldsymbol{p_1}$, the contact point on Object2, $\boldsymbol{p_2}$, the features on Object1 and Object2 that correspond to the closest feature pair, and a contact normal $\boldsymbol{n}$ pointing from Object2 to Object1. It also computes a distance value, d, based on the geometric features involved in the contact. The distance value is positive for disjoint objects and negative for penetrating objects.

4.2 Clustering Contacts

The convex surface decomposition algorithm can sometimes result in a number of small convex patches around concavities. This can lead to a large number of contacts in certain configurations. The stiffness of different contacts is added up to formulate a larger stiffness value, which may cause instability in the force feedback. Some of the techniques used for reducing the instability, caused by stiffness variations, limit the maximum total stiffness [McNeely et al. 1999]. However, these approaches do not account for the spatial distribution of the contacts and some contact areas can become stiffer than others, depending on the relative configuration of two objects.

To avoid instability problems due to large stiffness and irregular stiffness distributions, we group the contacts based on their proximity and distribution in the 3D space. In particular, we represent each contact with a single point $\hat{\boldsymbol{p}}$ by taking the average of the contact points, one from each object:

$$\hat{\boldsymbol{p}}_{\boldsymbol{i}} = (\boldsymbol{p_{1i}} + \boldsymbol{p_{2i}})/2 \tag{1}$$

Contacts that are closer than a threshold δ from each other are grouped and averaged to a single

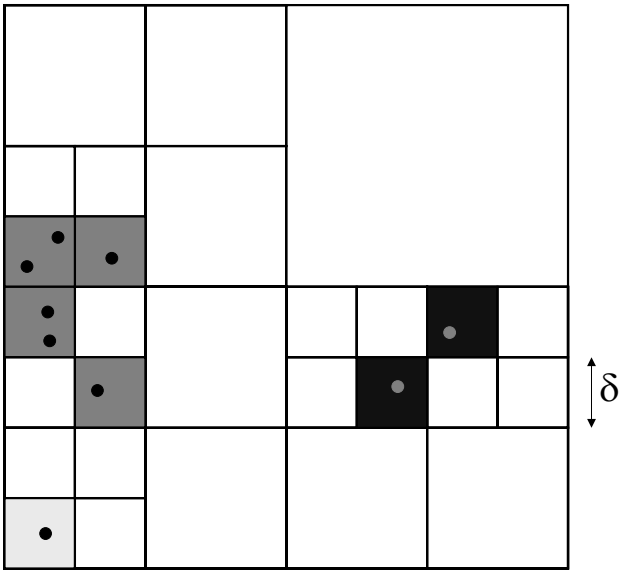


Figure 2: Contact Clustering Using Quadtrees. *The contacts are represented by dots. Different intensities are used to highlight the different clusters. Octrees are used instead of quadtrees in our 3D implementation.*

contact. Hence, given a contact cluster S and a contact c_j:

$$c_j \in S \iff \exists c_k \in S \text{ such that } \|\hat{\boldsymbol{p}}_{\boldsymbol{k}} - \hat{\boldsymbol{p}}_{\boldsymbol{j}}\| < \delta \tag{2}$$

We use an octree representation to cluster the contacts. First the contacts are inserted in the appropriate cell in the octree. The leaf cells are of size δ, as shown in Fig. 2. The clusters are initialized by grouping all the contacts in the same leaf cell. Next, adjacent non-empty clusters are merged together. Following this procedure, the clustering policy is uncertain if the distance between the contacts is in the range $(\delta, 2\sqrt{3}\delta)$. The range can be reduced by further subdividing the octree cells. Notice that, in order to maintain the haptic update rates, our clustering scheme approximates the clustering criterion given in Equation 2, as opposed to computing it exactly.

Every cluster is represented by one contact. It involves computing the position of the contact, the distance value, and a direction for the force corresponding to that particular contact. The position and the direction are computed in terms of a weighted average, where the weights are the distance values for each pair of contacts. Therefore, contacts with a deeper penetration contribute more to

the representative contact. This is computed as:

$$\boldsymbol{n} = \frac{\sum (t - d_i) \cdot \boldsymbol{n_i}}{\| \sum (t - d_i) \cdot \boldsymbol{n_i} \|} \tag{3}$$

$$\hat{\boldsymbol{p}} = \frac{\sum (t - d_i) \cdot \hat{\boldsymbol{p}}_{\boldsymbol{i}}}{\sum (t - d_i)} \tag{4}$$

where t is the distance tolerance for defining near contacts.

For computing the distance value, d, of the representative contact, we simply select the smallest of all contacts in the cluster (or the largest PD). That is,

$$d = \min(d_i) \tag{5}$$

4.3 Computation of the Force

For the force computation, we have chosen a force model that is computationally feasible in real-time and reproduces physically realistic forces as well. Essentially, our force model follows Hooke's law, where forces are proportional to displacements. Even though the force computation method based on the formulation of the linear complementarity problem [Baraff 1994] might reproduce higher-fidelity and more realistic contact forces, the solution turns out to be prohibitively slow for haptic rendering. In addition, Hooke's law can also approximate inter-object contact that results in very small deformation.

We compute an elastic force for each representative contact. The forces on Object1 and Object2 are computed as:

$$\boldsymbol{F_1} = (k \cdot (t - d) - k_v \cdot v) \cdot \boldsymbol{n} \tag{6}$$

$$\boldsymbol{F_2} = -\boldsymbol{F_1}, \tag{7}$$

where k and k_v are stiffness and damping constants, respectively.

In order to apply the damping force, we compute a relative velocity v at the representative contact by subtracting the velocities of both objects, $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$, at the point $\hat{\boldsymbol{p}}$ and projecting the result onto the contact direction $\boldsymbol{n}$:

$$v = (\boldsymbol{v_1} - \boldsymbol{v_2}) \cdot \boldsymbol{n} \tag{8}$$

To ensure the passivity, the damping force is applied only if the objects were approaching each other at the point of contact (i.e., when v is negative),

Moreover, torque is computed as a cross product of the vector from the center of mass of each object $\boldsymbol{o}$ to the contact point $\hat{\boldsymbol{p}}$ and the force $\boldsymbol{F}$:

$$\boldsymbol{T_1} = (\hat{\boldsymbol{p}} - \boldsymbol{o_1}) \times \boldsymbol{F_1} \tag{9}$$

$$\boldsymbol{T_2} = (\hat{\boldsymbol{p}} - \boldsymbol{o_2}) \times \boldsymbol{F_2} \tag{10}$$

The forces and torque are summed up for each object to compute the net force and torque. If the force and torque to be applied to the probe and to the user, exceed the maximum values of the haptic device, they are clamped to the maximum values, preserving the direction.

4.4 Additional Features

Our force computation model allows for implementing several additional features that can either yield more realistic forces or reduce instability problems. We can add per-contact friction forces, as well as corner rounding or force shading techniques.

4.4.1 Force Shading

We also use a force shading algorithm to reduce the discontinuities that happen when the contacts occur across some edge of the models [Morgenbesser and Srinivasan 1996]. In particular, we used the Gouraud shading algorithm for computing the normal direction at the contact points before clustering, based on per-vertex normals. Without this feature, vertices and edges appear as unstable points in the force computation, because the direction of the force changes abruptly when a contact transition is made from one feature to another.

4.4.2 Friction

Our force model also allows easy inclusion of friction models, such as the stick-slip type presented in [Hayward and Armstrong 2000]. In order to achieve this, we track the contacts in the time domain,

and define an adhesion point that allows for the computation of the friction forces. Unlike force shading, friction can be applied to the representative contacts of the clusters.

5 Penetration Depth Estimation

In this section, we present a new algorithm to estimate the penetration depth (PD) between polyhedral models. It is central to the penalty-based force computation algorithm described in Section 4 for calculating the restoring forces.

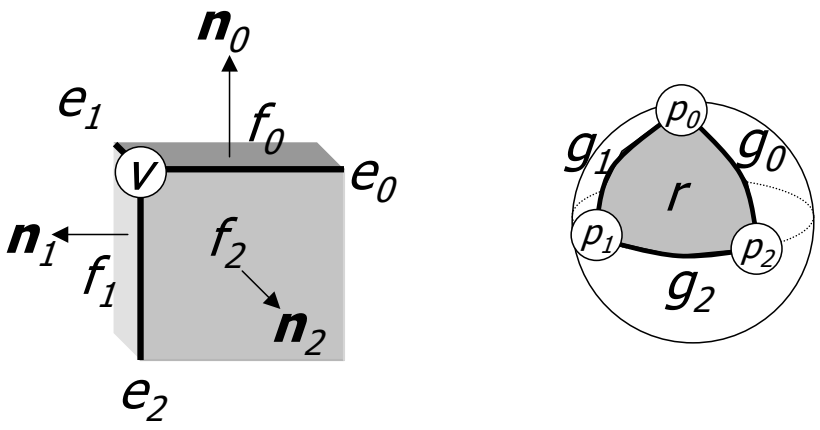
Given the overall complexity of exact penetration depth and real-time constraints of haptic rendering, we present a fast estimation algorithm that takes into account high coherence between successive frames and is relatively simple to implement. As part of a preprocess, the polyhedron is decomposed into convex solid pieces. At runtime, the algorithm computes the pairwise PD for each pair of overlapping convex pieces using an incremental method.

5.1 Preliminaries

We outline our notation and briefly define some of the terms used to design our PD estimation algorithm. We use a regular upper-case letter to denote a general feature (e.g. V, E, F for vertices, edges and faces, respectively) and use a italic lower-case letter to denote an instantiated particular feature (e.g. v_1, e_1, f_1). In particular, we use (V, V) to denote a *vertex hub pair* as will be explained in Section 5.2.

The Minkowski sum, $A \oplus B$, is defined as a set of pairwise sums of vectors from A and B. In other words, $A \oplus B = \{\boldsymbol{a} + \boldsymbol{b} | \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$. Similarly, the Configuration Space Obstacle (CSO) [Cameron 1997] or Minkowski sum, $A \oplus -B$ is defined as $\{\boldsymbol{a} - \boldsymbol{b} | \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$; see Fig. 4. Here, we use bold-faced letters to denote a vector quantity.

The Gauss map (or normal diagram) is a mapping from object space to the surface of a unit sphere $\mathbb{S}^2$ in 3D [de Carmo 1976]. In this mapping, a face and an edge are mapped to a point and a great arc on the sphere, respectively, and a vertex is mapped to a convex region. Thus, this mapping represents the mapping of features from the object space to the normal space; see Fig. 3.



(a) Features (b) Gauss Map

Figure 3: Gauss Map of a Polytope. *In (a), let us say that e_0, e_1, e_2 are the incident edges of a vertex v, and f_0, f_1, f_2 are the faces that share the edges e_0, e_1, e_2; the faces are also associated with its outward face normal n_0, n_1, n_2, respectively. In (b), the Gauss map for these features maps the faces f_0, f_1, f_2 to points p_0, p_1, p_2 on a unit sphere $\mathbb{S}^2$, respectively, the edges e_0, e_1, e_2 to great arcs r_0, r_1, r_2, and the vertex v to a convex region r.*

It is well known that once the Gauss map of two objects A and B and their overlay is computed, one can reconstruct $A \oplus B$ from the overlay. Moreover, only the vertex/face (VF), face/vertex (FV), and edge/edge (EE) pairs from each object contribute to the overlay [Guibas and Seidel 1987]. Therefore, when computing $A \oplus -B$, one needs to determine only VF, FV, and EE antipodal[2] pairs.

5.2 PD Estimation between Convex Polytopes

The PD between two objects is defined as the minimum translational distance to separate them. We first present an incremental PD estimation algorithm for convex polytopes and then extend it to general polyhedral models.

It is well known that the PD between objects A and B is the same as the minimum distance

[2]Since we negate one of the polytopes, $-B$, we need to reflect the Gauss map of B with respect to the origin.

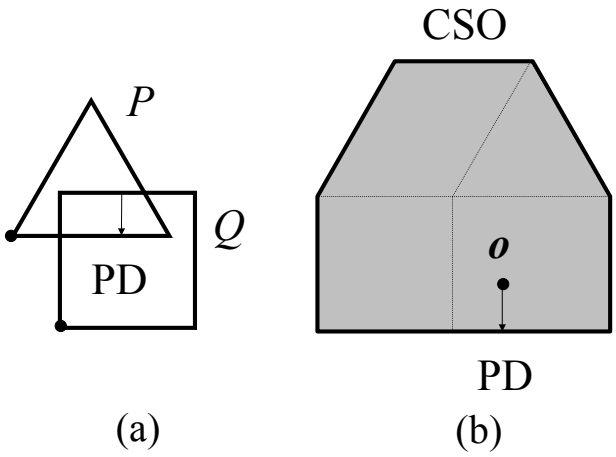


Figure 4: PD and CSO. *CSO is defined as $P \oplus -Q$, and PD corresponds to the minimum distance from the origin to the surface of CSO.*

from the difference vector $\boldsymbol{O}_{B-A}$ between the origins of A and B to the surface of Configuration Space Obstacles (CSO), or the Minkowski sum $A \oplus -B$ [Cameron 1997], see Fig. 4. Throughout the rest of the paper, the origin of the CSO will refer to the difference vector of the origins of two polyhedra, $\boldsymbol{O}_{B-A}$.

Our algorithm incrementally finds the "locally optimal" PD by locally constructing the surface of the CSO and walking on it (or the neighboring features). The local surface of the CSO is implicitly computed by constructing a local Gauss map. The "locally optimal" PD is defined using the features on the CSO. Let f be a feature on the CSO that realizes the locally optimal PD. Then, the distance from the origin to f is always smaller than the distance from the origin to any neighboring feature of f.

5.2.1 Local Optimization

Our algorithm incrementally computes the PD based on a local optimization technique. Starting from some feature(s) on the surface of the CSO, the algorithm finds the direction in which it can minimize the PD value, and proceeds to that direction by locally extending the surface of the CSO. Thus, the major computation step in the algorithm involves locally constructing the surface of the CSO and finding a good starting point to walk on the surface of CSO.

At each iteration of the algorithm, a vertex pair is chosen from each polytope. We refer to it as a *vertex hub pair*, and it serves as a hub of the expansion of the local CSO. The vertex hub pair is chosen in such a way that there exists a plane supporting each polytope and touching each vertex. Therefore, the regions corresponding to each vertex in the Gauss maps overlap. This intersection corresponds to the VF or EE antipodal pairs in the object space, from which one can reconstruct the local surface of the CSO around the vertex hub pair. Based on it we decide which antipodal pair provides the shortest distance from the origin of the CSO to the reconstructed local surface. If this pair decreases the estimated PD value, we update the current vertex hub pair. We iterate this procedure until we can not decrease the current PD estimate any more.

5.2.2 Initialization Step

The algorithm starts with an initial guess on the vertex hub pair ((V,V)) on each polytope (a region/region pair on the Gauss map). The initial guess is crucial for the performance of our incremental algorithm. A good initial guess can lead to "almost constant" running time, whereas a bad one can lead to $O(n^2)$ running time in the worst case, where n is the number of features in each polytope. There are many plausible strategies to pick a good initial guess. One simple approach involves taking the line joining the centroids of the objects, and find an extremal vertex pair along that direction. For instance, in Fig. 5-(a), c_1 and c_2 are the centroids of each object. The extremal vertex pair along the directions of $c_2 - c_1$ and $c_1 - c_2$ is chosen from each object, and is assigned as an initial vertex hub pair. This technique is known to suggest a good initial guess for the Voronoi Marching [Ehmann and Lin 2000], and also works well for PD estimation.

Other approaches to computing the initial vertex hub pair are based on the underlying contact determination algorithm. Whenever the objects penetrate, most contact determination algorithms report a witness feature pair [Cameron 1997; Ehmann and Lin 2000]. From this feature pair, one might be able to get closer to the actual PD feature pair, and thereby computing an optimal PD. One possibility is to consider the plane normal to the penetration feature as penetration direction and improve the estimate by local walking. For instance, in Fig. 5-(b), a face f is identified as a penetration witness, and its associated plane normal n is used for the extremal vertex query.

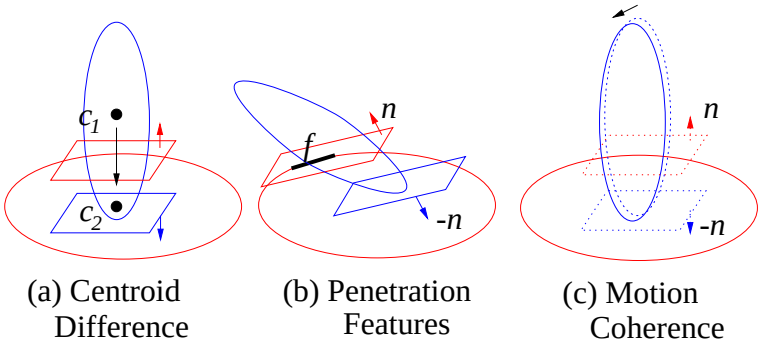


Figure 5: Various Initial Guessing Strategies. *(a) shows that the centroid difference vector $c_2 - c_1$ can approximate the penetration direction. (b) shows that the normal vector n of a penetration feature f approximates the penetration direction. (c) shows that the PD computation result (penetration vector n) from the previous time frame can provide the penetration direction.*

Given the high update rate of haptic display, there exists high coherence between successive frames. As a result, one can use the PD features or the pair of closest features from the previous frame as the initial guess for the current frame. In Fig. 5-(c), for example, the previous PD features provide a direction for the extremal vertex query.

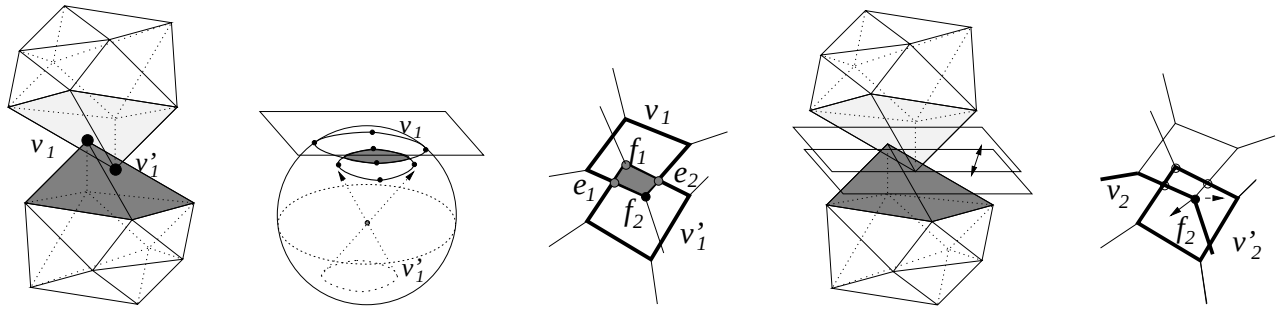
5.2.3 Iterative Optimization Step

After the algorithm obtains an initial guess on a (V,V) pair, it iteratively seeks a local improvement by jumping from one (V,V) pair to an adjacent (V,V) pair. This is accomplished by looking around the neighborhood of the current (V,V) pair and jumping to the pair that provides the greatest improvement in the PD value. In more detail, let's call the current vertex hub pair (v_1, v_1'). The next vertex hub pair (v_2, v_2') is computed as follows (as shown in Fig. 6):

1. Construct a local Gauss map each for v_1 and v_1'. See Fig. 6-(a) and 6-(b).

2. Project the Gauss maps onto $z = 1$ plane, and call them G and G' respectively. G and G' are convex polygons in 2D. See Fig. 6-(c).

3. Compute the intersection between G and G', and label each vertex comprising the intersection u_i. These u_i's correspond to the VF or EE antipodal pairs in object space. In Fig. 6-(b), f_1,



(a) (b) (c) (d) (e)

Figure 6: Iterative Optimization: *(a) The current (V,V) pair is* (v_1, v_1') *and a shaded region represents edges and faces incident to* (v_1, v_1')*. (b) shows local Gauss maps and their overlay for* (v_1, v_1')*. (c) shows the result of the overlay after central projection onto a plane. Here,* f_1*,* e_1*,* f_2 *and* e_2 *comprise vertices (candidate PD features) of the overlay. (d) illustrates how to compute the PD for the candidate PD features in object space. (e)* f_2 *is chosen as the next PD feature, thus* (v_2, v_2') *is determined as the next vertex hub pair.*

e_1, f_2, and e_2 are u_i's. $v_1 f_1$ and $f_2 v_2$ are VF and FV antipodal pairs and e_1 and e_2 are EE antipodal pairs.

4. In object space, compute u_i that provides the best local improvement in PD, and set an adjacent vertex pair to u_i to (v_2, v_2'). In Fig. 6-(c), f_2 is chosen and the corresponding PD is computed as in Fig. 6-(d). In Fig. 6-(e), v_2 is adjacent to f_2, thus (v_2, v_2') is chosen as a vertex hub pair for the next iteration.

This iteration is repeated until either there is no further improvement in the PD value or the iteration has reached some maximum number of iterations. At step 4 of the iteration, however, there are multiple choices for the vertex hub pair. If u_i corresponds to VF, then we must choose one of two vertices adjacent to u_i, assuming that the model is triangulated. The same reasoning also works when u_i corresponds to EE. Therefore we need one more iteration in order to actually decide which vertex hub pair should be checked. This extra iteration is reused at next iteration by caching the result and using it again. For example, in Fig. 6-(c), u_i is f_2, and there are two choices

for the direction to proceed. However, one more iteration suggests (v_2, v_2') as a next vertex hub pair.

5.3 Extension to Non-Convex Polyhedra

We do not attempt to compute one global PD between non-convex polyhedra, instead we compute a set of PD's for each pair of intersecting polytopes. Therefore, our force computation model computes the restoring force based on all contacts between all pairs of convex pieces. There are three reasons for taking this approach:

- The global PD computation can take $O(n^6)$ time in the worst case (based on the worst case complexity of the Minkowski sum or the CSO), where n is the number of features for each polyhedron. Even the fastest known algorithm to compute the PD between general polyhedral models [Kim et al. 2002b] can not keep up with the stringent time requirement of haptic updates.

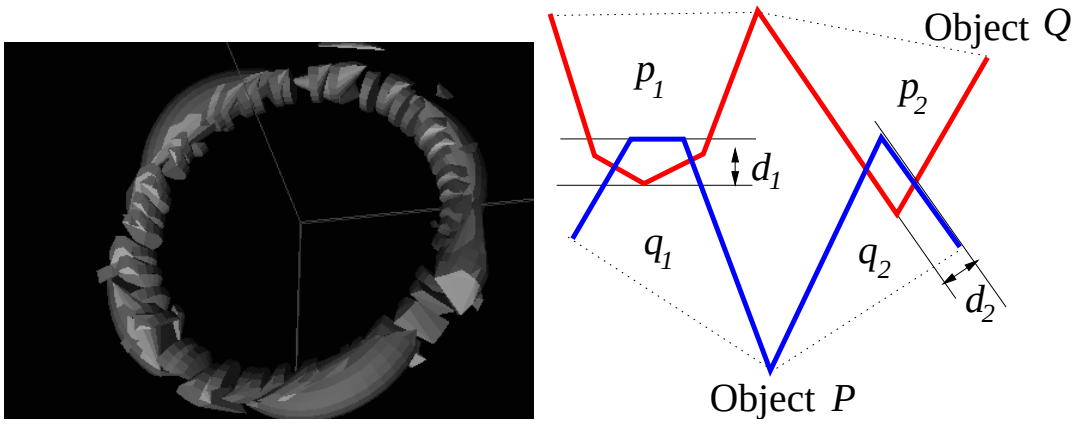
- The computation of multiple overlapping regions and PD's result in more stable force computation and transition (as explained in section 4).

- A set of pairwise PD's can compensate for the lack of consideration for rotational motion in the definition of PD. In this case, each PD element included in the PD set indicates a separate direction and magnitude for each local penetration situation, thus the combination of these PD's suggests plausible rotational motion used to separate the overlapping objects.

5.3.1 Convex Decomposition

In order to compute pairwise PD's between two general polyhedra, we decompose each polyhedron into a collection of convex pieces as a preprocessing step. We adopt the *convex surface decomposition* technique [Chazelle et al. 1997] because of the combinatorial simplicity in the convex decomposition.

We decompose the boundary of each (non-convex) polyhedron P into a collection of convex patches c_i. These c_i's are mutually disjoint except for their shared edges, and the union of all the c_i's covers the entire boundary of P, ∂P. We compute the convex patches, c_i's, by dualizing the



(a) A decomposed torus (b) Pairwise estimation

Figure 7: Extension to Non-Convex Objects

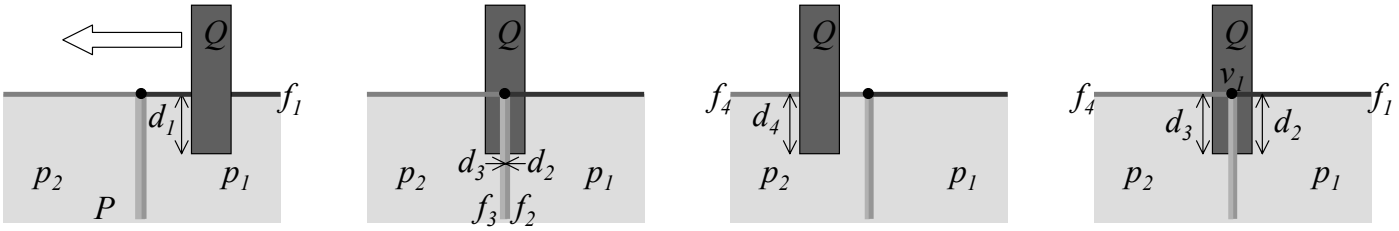
polyhedral surface and performing a graph search on it in a greedy manner. First, we construct a dual graph G of the polyhedral surface ∂P by reversing the roles of faces (F) and vertices (V) in ∂P, while using the same edges (E) in G from ∂P [Chazelle et al. 1997]. We traverse the dual graph G by adding faces into a current convex patch c_i, as long as it maintains its convexity. We repeat this process until we cover the entire boundary of ∂P. Furthermore, we compute a convex hull, C_i, of each surface patch, c_i. The union of these C_i's is completely contained in the original polyhedron P. For instance, Fig. 7-(a) shows an example of a convex decomposed torus model.

Notice that, during the convex hull construction from the convex patch c_i to the convex piece C_i, the algorithm can introduce some virtual features that do not exist in an original model. Since these virtual features are arbitrarily created depending on the traversal order in the dual graph G, we need to ensure that these faces are not be included in the PD results. This will be explained in more detail in the next section.

5.3.2 Pairwise PD Computation

At runtime, we perform the pairwise PD computation as follows:

1. Identify a set of pairwise overlapping convex pieces, $\mathcal{C} = \{< p_0, q_0 >, < p_1, q_1 >, \cdots, < p_n, q_n >\}$, between two overlapping polyhedra, P and Q.



(a) (b) (c) (d)

Figure 8: Discontinuity in pairwise PD computations. *A convex object Q penetrates into a non-convex object P (decomposed into p_1, p_2), and moves to the left. In (a), (b), (c), the pairwise PD results are $d_1, \{d_2, d_3\}, d_4$, respectively, and their associated PD features in P are $f_1, \{f_2, f_3\}, f_4$. However, in (b), f_2 and f_3 are non-original features, thus they should not be considered for PD results. This makes a jerky transition from (a) to (c). In (d), we solve this problem by promoting the PD features to nearby original features v_1 and finding the next best PD results from there. Moreover, contact clustering will group the two contacts computed in (d).*

2. For each $< p_i, q_i >$ in $\mathcal{C}$, we compute its PD, d_i, using the algorithm presented in Section 5.2.

For instance, in Fig. 7-(b), a non-convex object P is decomposed into two convex pieces p_1 and p_2, and another non-convex object Q is decomposed into q_1 and q_2. For each intersecting convex pieces, p_1 and q_1, and p_2 and q_2, we compute a set of PD's, d_1 and d_2, both of which are used in our force computation algorithm.

Earlier in Section 5.3.1, we observed that our convex surface decomposition algorithm introduces some non-original or "virtual" features. Therefore the result of the PD computation can contain features that should not be considered for force computation. Simply ignoring these virtual features can cause transition problems in our force computation model. For example, in Fig. 8, a convex object Q penetrates into a non-convex object P which is decomposed into convex pieces p_1 and p_2, and Q moves to the left as illustrated in Fig. 8-(a) $\sim$ (c). In Fig. 8-(a), there is only one overlapping pairs $< p_1, Q >$, and its associated PD is d_1. Since the PD feature f_1 in P that realizes the PD d_1 is an original feature in P, d_1 is reported as an legitimate PD value. However, when Q

moves to the left as in Fig. 8-(b), for both of the overlapping pairs, $< p_1, Q >$ and $< p_2, Q >$, their associated PD features f_2 and f_3 correspond to virtual features. In this case, if we ignore these PD results, there will be no features available for the force computation algorithm. When Q reaches the position shown in Fig. 8-(c), for the overlapping pair $< p_2, Q >$, we have a legitimate PD, d_4, and the corresponding features f_4, since f_4 is an original feature in P. Therefore, as the object Q moves from (a) to (c), we would experience the jerky transition of PD values and their associated PD features, even though the surface characteristics of the object P is seemingly smooth.

One possible way to address this problem is to find the next best (or closest) original features on each convex polytope (i.e. the features that belong to the original non-convex polyhedra). In our algorithm, we take advantage of the fact that (a) vertices are never created nor removed by our convex decomposition scheme as explained in Section 5.3.1 (i.e. the vertices are always original), and (b) a vertex is always adjacent to some original features. Therefore, once we realize that the reported PD features are virtual (non-original), we first find the adjacent vertices to the virtual PD features, then apply one iteration of the PD computation (using them as the features) to compute features on the original polyhedra that correspond to locally optimal values. For instance, as opposed to ignoring the virtual PD features f_3, f_4 as in Fig. 8-(b), we find their adjacent vertex, v_1[3]. Then, we choose v_1 as one element of the vertex hub pair (V,V), and apply one iteration to the pair. As a result, the new PD values are d_2, d_3 and their associated original PD features are f_1, f_4, respectively. Meanwhile, if restoring forces were computed at all contacts, the transition from (a) to (d) and to (c) would still not be stable, because a change in the amount of contacts would bring a change in the perceived force. We solve this problem with the contact clustering technique described in Section 4.2. Thus, we can expect a smooth transition of PD results from (a) to (d), and to (c).

[3]In this example, v_1 happens to be shared by f_3 and f_4. But it needs not to be always this shared case.

Model	Convex Pieces	Triangles
Torus	67	2000
Cup	190	500
Spoon	78	336
Hammer	425	1692
Armadillo	9098	31240
Digestive System	9913	24846

Table 1: Complexity of some of the models used in our benchmarking.

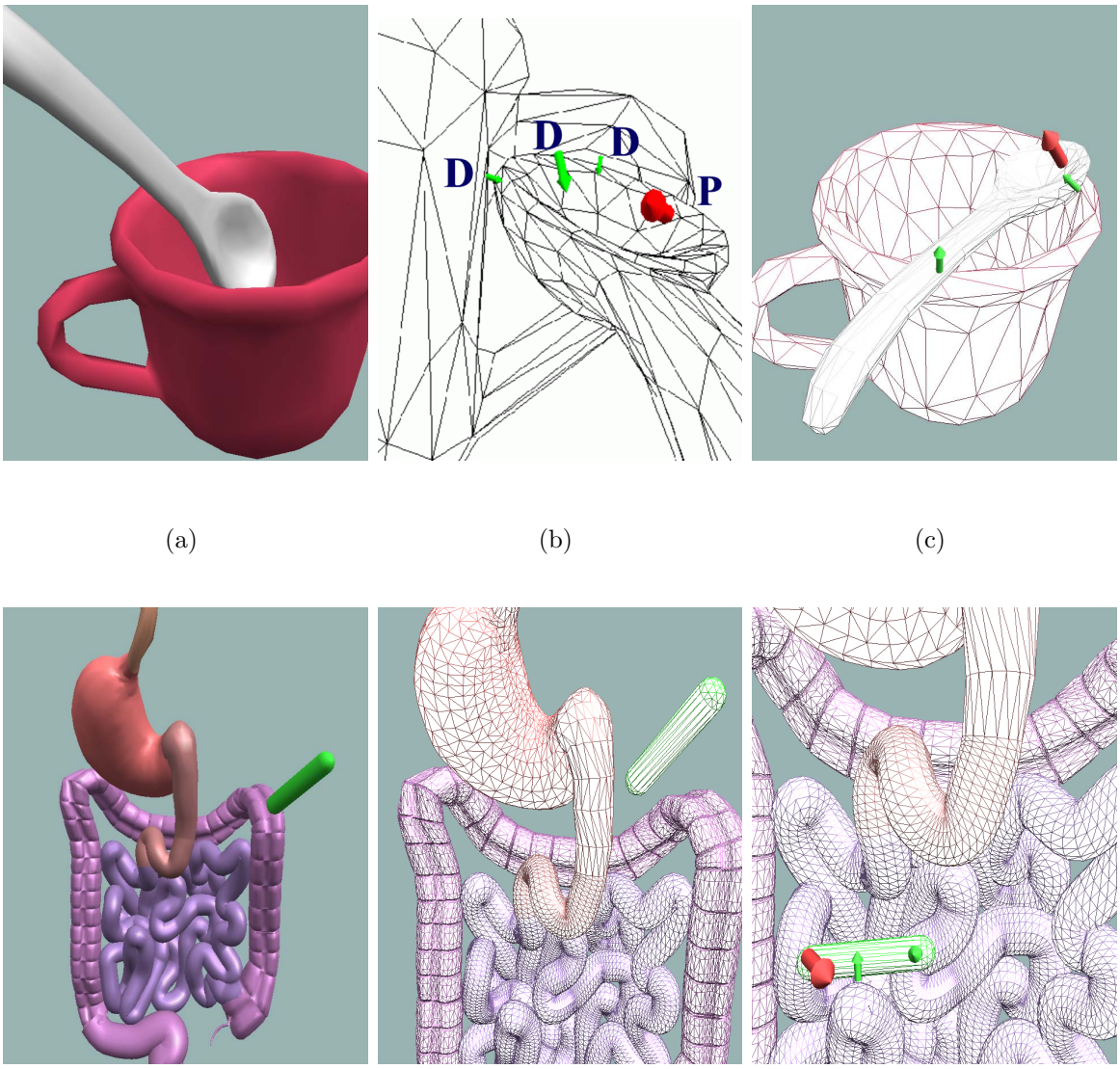
6 Implementation and Results

We have implemented the algorithms described in this paper and integrated them with force feedback hardware. In this section, we demonstrate the results of our 6-DOF haptic display framework.

6.1 Experiment Description

We performed our experiments on a Windows 2000 PC with dual 1-GHz Pentium III CPUs and 500 MB memory with a 6-DOF PHANToM Premium 1.5 haptic device.

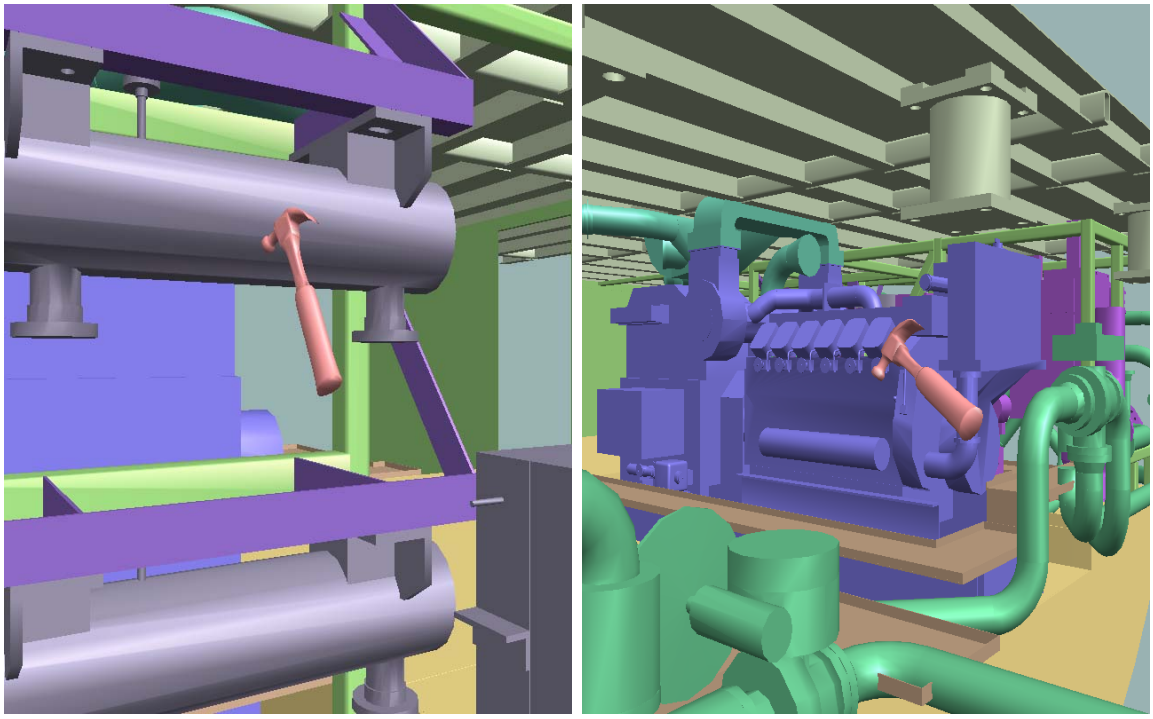
Table 1 highlights some of the models that we have used to test the performance of our prototype implementation. Figures 9 and 10 illustrate typical benchmark scenarios using our haptic simulation framework. In Figure 9-(a) $\sim$ (c), a non-convex object, a spoon, is touching the surface of another non-convex object, a cup. In the particular configuration shown in Fig. 9-(b), the contacts returned by the collision detection module are clustered in four groups. As a result, we have one contact (P) due to penetration and three other contacts that are within the tolerance threshold (D). The arrows in the figure denote the direction of the resulting restoring forces, and their sizes denote the amount of the forces. Figure 9-(d) $\sim$ (f) shows another haptic simulation where a convex probe is touching a non-convex object, a digestive system model. In Figure 10 we show two snapshots of a virtual prototyping application. A tool is haptically interacting with the model of the auxiliary machine room (AMR) of a submarine, consisting of 897 objects with the triangle count of $195,926$.



(a) (b) (c)

(d) (e) (f)

Figure 9: Two Contact Scenarios: a Cup interacting with a Spoon (Top) and Virtual Exploration of a Digestive System Model (Bottom). *Figures in (b), (c), (e) and (f) show the forces computed at clustered contacts, for both disjoint (D, green arrows) and penetrating (P, red arrow) situations.*



(a) (b)

Figure 10: Two Contact Scenarios in Virtual Prototyping of AMR Models.

The tool comes into contact with 17 objects that consist of $3,810$ triangles.

6.2 Results

Figure 11-(a) illustrates a typical timing profile of a cup interacting with a spoon scenario as shown in Figure 9-(a). Figure 11-(b) shows a profile of a probe interacting with a digestive system model as in Figure 9-(d). Finally, Figure 11-(c) shows a timing profile of a hammer interacting with the AMR model as in Figure 10. In Figure 11-(a) $\sim$ (c), the lower chart shows the number of contacts reported by the contact query, while the upper chart, from top to bottom respectively, shows the time consumed by the contact query (the sum of tolerance-based collision detection and PD computation) and the time needed for force computation. The charts are divided into several intervals, depending on whether objects are disjoint (D) or inter-penetrating (P). The charts show that even in a complex interaction scenario as the cup and the spoon having up to 30 contacts at

a given time, the haptic simulation was able to proceed at a frequency of 500 Hz. Note that the presence of noise in the upper chart is due to the scheduling problem inherent in the underlying operating system. Also, note that our algorithm for estimating PD only introduces a very small amount of overhead to the time spent on the collision detection.

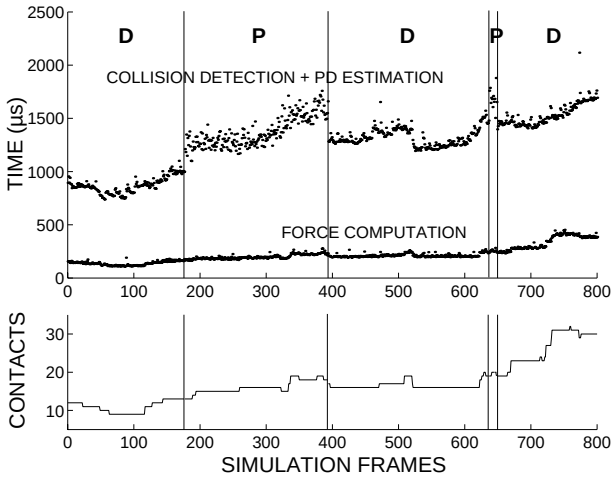
Figures 12-(a), -(b), and -(c) track the computed results on a contact point, the corresponding contact normal, and the distance value respectively between two convex objects. The data has been recorded by traversing a sphere with a pen-like object. Since both of them are convex, there is only one contact, and this allows a better study of the behavior of our algorithm. In the figures, the thin solid curve represents a disjoint situation, whereas the thick solid curve indicates that the objects are inter-penetrating. In 12-(c), the negative distance values also mean that the objects are inter-penetrating. One can notice the smooth transition of the contact information between disjoint and inter-penetrating situations, as well as the smooth evolution during long penetrating paths. This enables our haptic rendering framework to display both smooth force and torque to an end user, without introducing any discontinuity that often incurred by existing PD computation methods.

6.3 Analysis

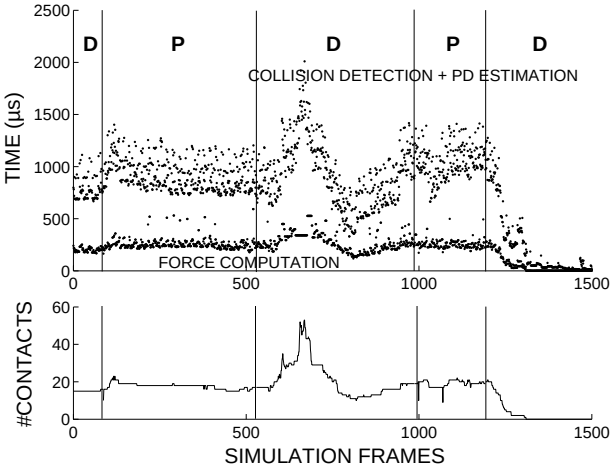
The time spent by the haptic simulation depends on the complexity of the models as well as on the contact scenario. In general, the time for the contact determination increases roughly proportionally to the number of contacts. The main reason is that as the number of contacts increases, the algorithm needs to traverse more nodes in the BVH. When objects inter-penetrate, the PD computation adds a small increase to the contact query time, as can be noticed from Figure 11.

More specifically, the PD computation time was linear in terms of the number of intersected pairs of convex pieces, since the computation is almost constant regardless of the triangle counts of each convex object. We refer readers to see [Kim et al. 2002a] for extensive experimental results on our PD computation and its implementation, DEEP[4]. Roughly, the PD computation time takes about

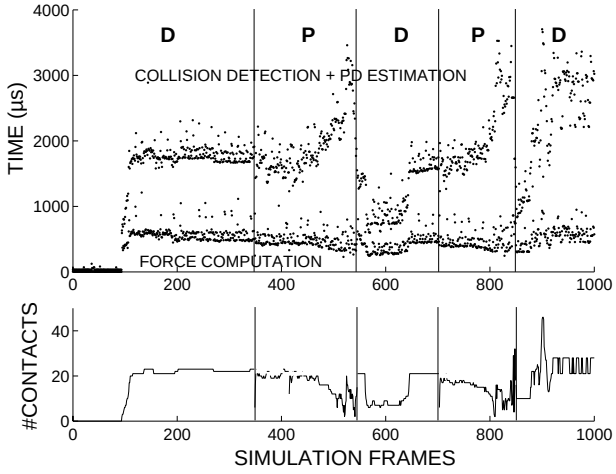
[4]Dual-space Expansion for Estimating Penetration depth



(a) Cup and Spoon

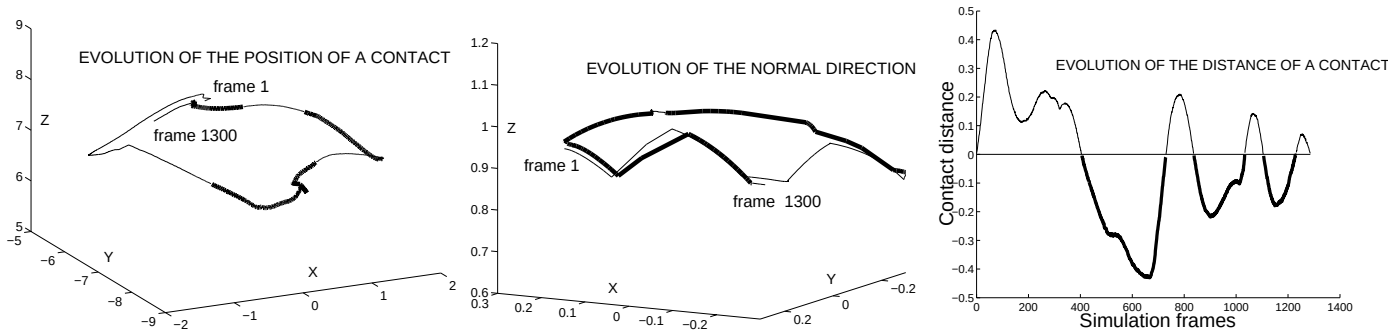


(b) Digestive System



(c) AMR

Figure 11: Computation Profile. *Recorded while (a) the cup model touches the spoon model, (b) the probe touches the digestive system model and (c) the hammer explores the AMR. In each figure, the upper chart denotes the computation time for the contact query and the force computation, and the lower chart denotes the number of contacts before clustering. Vertical lines indicate the intervals of disjoint (D) and inter-penetrating (P) contact. Note that our PD estimation only introduces a very small amount of overhead to the time spent on the collision detection.*



(a) Contact Point (b) Contact Normal (c) Inter-distance

Figure 12: Contact Profile for a Single Contact Scenario. *The thick solid line represents that the objects inter-penetrate, and the thin solid line indicates that the objects are disjoint. Notice the smoothness of the contact data.*

0.1 msec, on average, using a single 1 GHz Pentium III CPU and Linux operating system, regardless of the complexity of objects.

The complexity of force computation after contact clustering is sub-linear in the number of original contacts returned by the contact query. This is due to the fact that the clustering operation significantly reduces the number of pairwise contact force computations.

The haptic simulation runs at higher frequency (greater than KHz) for less challenging scenarios (with few contacts), and slows down slightly with highly complex models such as an armadillo model or a digestive system model (see Table 1). We are currently working on an optimized implementation of our contact query method to handle highly complex models more efficiently.

Compared to previous approaches on 6-DOF haptic rendering, our algorithm offers improvement in several aspects. It is able to handle penetration computations more reliably and accurately as compared to earlier approaches. In [Gregory et al. 2000] penetration depth is roughly estimated along the direction of motion using the previous closest feature pairs, whereas our algorithm computes a locally optimal penetration value, which often turns out to be the exact penetration depth in most scenarios. This ensures more stable and realistic force computation. In addition, our method avoids artifacts such as force discontinuity arising from discretization problems inherent to existing

volumetric approaches.

7 Conclusion

We presented a 6-DOF haptic display algorithm with localized contact query and force computation methods for polyhedral models. In order to achieve the desired force update rates for haptic simulation, we employ incremental algorithms for contact queries by exploiting spatial and temporal coherence, and cluster contacts in a localized neighborhood to improve stability of force computation. Our algorithmic framework has been tested on models of varying complexity and worked well on different challenging scenarios.

There are several possible future research directions. We are currently investigating other approaches (e.g. multi-resolution techniques, coherence-based methods, etc) to design more robust and general algorithms to handle non-convex objects. We are also considering other algorithms to achieve faster PD computation.

References

Adachi, Y., Kumano, T., and Ogino, K. 1995. Intermediate representation for stiff virtual objects. In *Proc. IEEE Virtual Reality Annual Intl. Symposium*, 203–210.

Adams, R. J., and Hannaford, B. 1998. A two-port framework for the design of unconditionally stable haptic interfaces. In *Intl. Conference on Intelligent Robots and Systems*.

Agarwal, P., Guibas, L. J., Har-Peled, S., Rabinovitch, A., and Sharir, M. 2000. Penetration depth of two convex polytopes in 3D. *Nordic J. Computing 7*, 227–240.

Baraff, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of SIGGRAPH '94*, A. Glassner, Ed., ACM SIGGRAPH, 23–34. ISBN 0-89791-667-0.

Basdogan, C., Ho, C.-H., and Srinivasan, M. A. 1997. A ray-based haptic rendering technique for displaying shape and texture of 3D objects in virtual environments. *DSC-Vol. 61, Proceedings of the ASME Dynamic Systems and Control Division*, pp. 77–84.

Beckmann, N., Kriegel, H., Schneider, R., and Seeger, B. 1990. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. on Management of Data*, 322–331.

Bergen, G. 2001. Proximity queries and penetration depth computation on 3D game objects. *Game Developers Conference*.

Burdea, G. 1996. *Force and Touch Feedback for Virtual Reality*. John Wiley and Sons.

Cameron, S. 1997. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, 3112–3117.

Chazelle, B., Dobkin, D., Shouraboura, N., and Tal, A. 1997. Strategies for polyhedral surface decomposition: An experimental study. *Comput. Geom. Theory Appl. 7*, 327–342.

Colgate, J. E., Stanley, M. C., and Brown, J. M. 1995. Issues in the haptic display of tool use. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 140–145.

de Carmo, M. 1976. *Differential Geometry of Curves and Surfaces*. Prentice Hall, Englewood Cliffs, NJ.

Dobkin, D., Hershberger, J., Kirkpatrick, D., and Suri, S. 1993. Computing the intersection-depth of polyhedra. *Algorithmica 9*, 518–533.

Ehmann, S., and Lin, M. C. 2000. Accelerated proximity queries between convex polyhedra using multi-level Voronoi marching. *Proc. of IROS*.

Ehmann, S., and Lin, M. 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Proc. Eurographics*.

Fisher, S., and Lin, M. 2001. Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proc. International Conf. on Intelligent Robots and Systems*.

Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. 1988. A fast procedure for computing the distance between objects in three-dimensional space. *IEEE J. Robotics and Automation vol RA-4*, 193–203.

Gottschalk, S., Lin, M., and Manocha, D. 1996. OBB-Tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, 171–180.

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

Gregory, A., Mascarenhas, A., Ehmann, S., Lin, M. C., and Manocha, D. 2000. 6-DOF haptic display of polygonal models. *Proc. of IEEE Visualization Conference*.

Guibas, L. J., and Seidel, R. 1987. Computing convolutions by reciprocal search. *Discrete Comput. Geom. 2*, 175–193.

Guibas, L., Hsu, D., and Zhang, L. 1999. *H-Walk*: Hierarchical distance computation for moving convex bodies. *Proc. of ACM Symposium on Computational Geometry*.

Hayward, V., and Armstrong, B. 2000. A new computational model of friction applied to haptic rendering. In *Experimental Robotics VI*, Springer-Verlag, New York, P. Cork and J. Trevelyan, Eds., vol. 250, 404–412.

Ho, C.-H., Basdogan, C., and Srinivasan, M. A. 1999. Efficient point-based rendering techniques for haptic display of virtual objects. *Presence 8*, 5, pp. 477–491.

Hoff, K., Zaferakis, A., Lin, M., and Manocha, D. 2001. Fast and simple geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*.

Hubbard, P. M. 1995. Collision detection for interactive graphics applications. *IEEE Trans. Visualization and Computer Graphics 1*, 3 (Sept.), 218–230.

Kim, Y. J., Lin, M. C., and Manocha, D. 2002. DEEP: Dual-space Expansion for Estimating Penetration depth between convex polytopes. In *IEEE Conference on Robotics and Automation*.

Kim, Y. J., Otaduy, M. A., Lin, M. C., and Manocha, D. 2002. Fast penetration depth computation for physically-based animation. In *ACM Symposium on Computer Animation*.

Klosowski, J., Held, M., Mitchell, J. S. B., Zikan, K., and Sowizral, H. 1998. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Trans. Visualizat. Comput. Graph. 4*, 1, 21–36.

Larsen, E., Gottschalk, S., Lin, M., and Manocha, D. 1999. Fast proximity queries with swept sphere volumes. Tech. Rep. TR99-018, Department of Computer Science, University of North Carolina.

Lin, M., and Canny, J. F. 1991. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, 1008–1014.

Mark, W., Randolph, S., Finch, M., Van Verth, J., and Taylor II, R. M. 1996. Adding force feedback to graphics systems: Issues and solutions. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 447–452.

Massie, T. M., and Salisbury, J. K. 1994. The phantom haptic interface: A device for probing virtual objects. *Proc. of ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1*, 295–301.

McKenna, M., and Zeltzer, D. 1990. Dynamic simulation of autonomous legged locomotion. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, F. Baskett, Ed., vol. 24, 29–38.

McNeely, W., Puterbaugh, K., and Troy, J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. *Proc. of ACM SIGGRAPH*, 401–408.

Mirtich, B. 1998. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics 17*, 3 (July), 177–208.

Moore, M., and Wilhelms, J. 1988. Collision detection and response for computer animation. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, J. Dill, Ed., vol. 22, 289–298.

Morgenbesser, H. B., and Srinivasan, M. A. 1996. Force shading for haptic perception. In *ASME International Mechanical Engineering Congress and Exposition*, vol. 58, 407–412.

Nahvi, A., Nelson, D., Hollerbach, J., and Johnson, D. 1998. Haptic manipulation of virtual mechanisms from mechanical CAD designs. In *Proc. of IEEE Conference on Robotics and Automation*, 375–380.

Ruspini, D., Kolarov, K., and Khatib, O. 1997. The haptic display of complex graphical environments. *Proc. of ACM SIGGRAPH*, 345–352.

Seidel, R. 1990. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, 211–215.

SensAble Technologies, I. 1997. *ghost*^{TM}: Sofware developer's toolkit. *Programmer's Guide*.

Thompson, T. V., Johnson, D., and Cohen, E. 1997. Direct haptic rendering of sculptured models. *Proceedings of ACM Interactive 3D Graphics*, pp. 167–176.

Zilles, C., and Salisbury, K. 1995. A constraint-based god object method for haptics display. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robotics and Systems*.