Six-Degree-of-Freedom Haptic Display Using Localized Contact Computations[*]

Young J. Kim Miguel A. Otaduy Ming C. Lin and Dinesh Manocha
Department of Computer Science
University of North Carolina in Chapel Hill
{youngkim,otaduy,lin,dm}@cs.unc.edu

Abstract: *We present a six-degree-of-freedom haptic rendering algorithm using localized contact computations. It takes advantage of high motion coherence due to fast force update and spatial locality near the contact regions. We first decompose the surface of each polyhedron into convex pieces and construct bounding volume hierarchies for fast proximity queries. Once the objects are intersecting, the penetration depth (PD) is estimated in the contact neighborhood between each pair of decomposed convex pieces, using a new incremental method based on local optimization. Given the computed PD values, multiple contacts near a local region are clustered together to further speed up contact force determination. We have implemented these algorithms and applied them to complex contact scenarios consisting of multiple contacts.*

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

1 Introduction

Intelligent systems and simulated environments 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) [7].

Some of the commercially available and commonly used haptic devices include the 3-degree-of-freedom (3-DOF) PHAMToM arm [28], SARCOS Dexterous Arm [34] 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 in the desired force updates (typically at 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 [15, 22, 35, 37] and are not directly applicable for accurately 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. The only known algorithms based on computing the Minkowski difference of the polyhedra can have $O(n^6)$ worst case complexity, where n is the number of features [9]. Some of the proposed solutions for 6-DOF haptic rendering use either voxel sampling [30] or prediction methods [16] to compute the response force, which can result in force discontinuities or unstable behavior. Besides haptic rendering, the penetration depth computation problem also arises in dynamic simulation and robot motion planning.

1.1 Main Results

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

- 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 difference, 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. We decompose each non-convex polyhedron into a collection of convex pieces and compute localized PD between overlapping convex polytopes.
- A localized force model by clustering nearby surface contacts to reduce the computational costs in force display and improve the overall stability of force updates. This force model is built upon prediction methods and PD estimation,

[*]Supported in part by ARO Contract DAAG55-98-1-0322, DOE ASCII Grant, NSF Grants NSG-9876914, NSF DMI-9900157 and NSF IIS-982167, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, Intel and a fellowship of the Government of the Basque Country

and can easily include force shading and various friction models.

These approaches take advantage of motion coherence due to the high force updates of the haptic display and spatial locality around the neighborhood of the contact regions, to dramatically speed up PD calculations and contact force computations between general polytopes. We have successfully demonstrated our 6-DOF haptic rendering algorithm on several benchmarks and complex contact scenarios. 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. Section 3 describes a novel penetration depth algorithm. Section 4 presents our force computation model based on this penetration depth estimation method. Section 5 describes the system implementation and demonstrates the effectiveness of our algorithm.

2 Previous Work

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

2.1 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 [36], incremental computation of Minkowski difference [8, 13], feature tracking based on Voronoi regions [26, 31] and multi-resolution methods [11, 17]. Some of these algorithms are based on incremental computations and exploit frame-to-frame coherence [8, 26, 31].

For general polygonal models, bounding volume hierarchies (BVHs) 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 [5], OBB trees [14], sphere trees [21], k-dops [24], Swept Sphere Volumes [25], and convex hull-based trees [10].

2.2 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 [9]. 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. 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 [3]. 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 [8]. This has been further extended by Bergen, who improves the PD estimate by expanding a polyhedral approximation of the Minkowski difference of two polytopes [6].

Other approximation approaches are based on discretized distance fields. These include an approach based on graphics rasterization hardware and multi-pass rendering approaches [20] as well as the use of fast marching level-set algorithms [12].

2.3 Contact Force Computation

Different techniques have been proposed for rigid body dynamics and contact force computation. Usually the choice of the method used depends on the application requirements. Constraint-based dynamics computes contact response based on the formulation of a linear complementarity problem (LCP) [4]. Although this approach computes 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 [29, 32]. Penalty forces are usually computed as elastic forces which depend on the interpenetration between 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. Braking forces [30], which have a viscous nature, can also be considered as an example in this category.

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. Forces can be computed using constraint-based techniques such as the god-object [38] and virtual proxy [35], as well as intermediate representations like the intermediate plane [1, 27].

For 6-DOF haptic rendering, both volumetric approaches [30] and prediction methods for polygonal models [16] 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 [2]. Our algorithm uses the same collision detection algorithm as described in [16]. However, the haptic rendering presented in [16] does not compute a good estimate of penetration depth for its penalty-based contact force model and instability can often arise within the concavity regions.

3 Penetration Depth Estimation

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

Given the overall complexity of exact penetration depth and real-time constraints of haptic rendering, we develop a fast estimation algorithm that takes into account high coherence between successive frames and is relatively simple to implement. We decompose the boundary of each polyhedron into convex surface patches that are used to form convex solid pieces. We then compute the pairwise PD for each pair of overlapping convex pieces using an incremental method.

3.1 Preliminaries

In this section, 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). We also use a bold-faced letter to denote a *vertex hub pair* as will be explained later.

The Minkowski sum, $A \oplus B$, is defined as a set of pairwise sums of vectors from A and B. In other words, $A \oplus B = \{a + b | a \in A, b \in B\}$. Similarly, the Configuration Space Obstacle (CSO) [8] or Minkowski difference, $A \oplus -B$ is defined as $\{a - b | a \in A, b \in B\}$.

The Gauss map (or normal diagram) is a mapping from object space to the surface of a unit sphere in 3D. 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.

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 the overlay [18]. Therefore, when computing $A \oplus -B$, one needs to determine only VF, FV, and EE antipodal[1] pairs.

3.2 Intersection (Collision) Detection

The algorithm initially checks whether two objects are overlapping or disjoint. If objects are disjoint, we determine the closest features between them along with their associated distance measures. The features may correspond to a vertex (V), edge (E) or a face (F). Otherwise, we identify the intersection regions and estimate the penetration depth (PD) and associated PD features[2].

We use a convex surface decomposition based on a variant of breadth-first-search and bounding volume hierarchies (BVH) to handle both disjoint and intersected cases uniformly. We decompose the surface of each non-convex polyhedron into a collection of convex patches. Convex pieces are then formed by taking the convex hull of each surface patch. We build a BVH for each object out of convex pieces. In our BVH scheme, the root bounding volume (BV) corresponds to the convex hull of an object. For more detailed discussion on the hierarchy and convex surface decomposition, please refer to [10].

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 [26] 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 hierarchies until there is no intersection between the leaf nodes or until there is no leaf node within some tolerance value.

3.3 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.

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

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

It is well known that the PD between objects A and B is the same as the minimum distance from the origin of the Configuration Space Obstacles (CSO), or the Minkowski difference $A \oplus -B$, to its surface [8]. 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 which 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 to f.

3.3.1 Local Optimization

Our algorithm incrementally computes the PD based on local optimization. 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 for the walk on the 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.

3.3.2 Initialization Step

The algorithm starts with an initial guess on the vertex hub pair (**VV**) 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 polyhedron. 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. 1-(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 [11], 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 [8, 11]. From this feature pair, one might be able to get closer to the actual PD feature pair, and thereby computing an optimal PD. One possible way is to consider the plane normal to the penetration feature as penetration direction and improve the estimate by local walking. For instance, in Fig. 1-(b), a face f is identified as a penetration witness, and its associated plane normal n is used for the extremal vertex query.

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

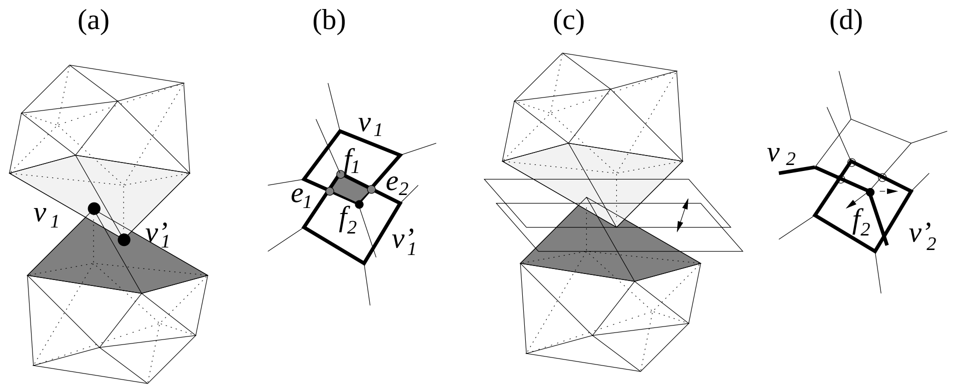


Figure 2. Iterative Optimization. (a) shows that the current VV pair is $v_1v'_1$. (b) shows the Gauss maps for $v_1v'_1$ after the central projection onto a plane. (c) shows how to compute the PD for the PD candidate features, f_1, e_1, f_2, e_2, in object space. (d) shows that f_2 is chosen as the next PD feature, thus $v_2v'_2$ is determined as the next vertex hub pair.

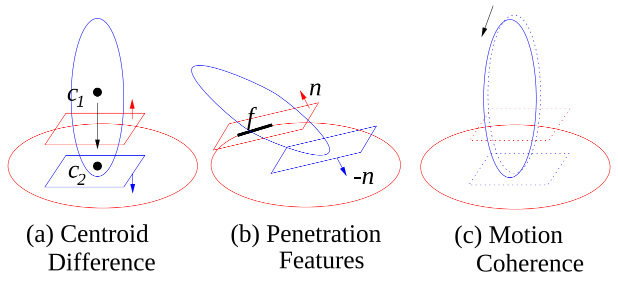


Figure 1. Various Initial Guessing Strategies.

as the initial guess for the current frame. In Fig. 1-(c), for example, the previous PD features provide a direction for the extremal vertex query.

3.3.3 Iterative Optimization Step

After the algorithm obtains an initial guess on a **VV** pair, it iteratively seeks a local improvement by jumping from one **VV** pair to an adjacent **VV** pair. This is accomplished by looking around the neighborhood of the current **VV** pair and jumping to the pair which provides a greatest improvement in the PD value. In more detail, let's call the current vertex hub pair $\boldsymbol{v_1v'_1}$. The next vertex hub pair $\boldsymbol{v_2v'_2}$ is computed as follows (as shown in Fig. 2):

1. Construct a local Gauss map each for v_1 and v'_1. See Fig. 2-(a) and 2-(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. 2-(b).
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. 2-(b), f_1, e_1, f_2, and e_2 are u_i's. v_1f_1 and f_2v_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 $\boldsymbol{v_2v'_2}$. In Fig. 2-(b), f_2 is chosen and the corresponding PD is computed as in Fig. 2-(c). In Fig. 2-(d), v_2 is adjacent to f_2, thus $\boldsymbol{v_2v'_2}$ 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. 2-(c), u_i is f_2, and there are two choices for the direction to proceed. However, one more iteration suggests $\boldsymbol{v_2v'_2}$ as a next vertex hub pair.

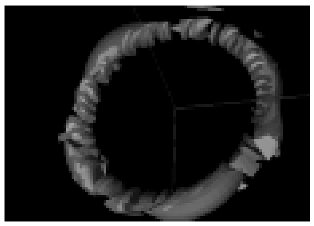
3.4 Extension to Non-Convex Polyhedra

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 two reasons for taking this approach:

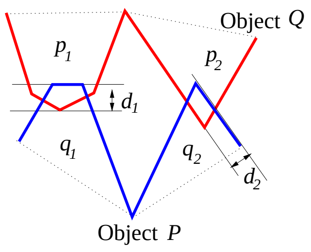
- The only known algorithm for global PD computation takes $O(n^6)$ time in the worst case (based on the worst case complexity of the Minkowski sum or the CSO), where n is the number of features for each polyhedron. No implementations are known of this algorithm.
- The computation of multiple overlapping regions and PD's result in more stable force computation and transition (to be explained in section 4).

For instance, Fig. 3-(a) shows an example of a convex decomposed torus model. In Fig. 3-(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. A single global estimation for PD in such cases can lead to



(a) A decomposed torus



(b) Pairwise estimation

Figure 3. Extension to Non-Convex Objects

a jerky transition of features when objects move from one overlapping convex pair to the other.

In the meantime, 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 also cause transition problem sometimes. One possible way to address the problems 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 first find the adjacent features to be the virtual PD features, and then apply one iteration of the PD computation to them to compute the locally optimal values.

4 Force Computation Model

In this section we describe how the restoring forces are computed, given the localized PDs 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 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. However, computation of exact contact for haptic simulation can be too expensive for even simple contact scenarios. Therefore, we define a "contact" between two objects as a pair of features, one from each object, that are within proximity of a certain user-defined tolerance. For disjoint cases when two objects are nearly in contact yet separated, we could describe the contact regions as the volume within a predefined distance tolerance. On the other hand, for penetrating contacts, we would consider the actual penetration volume. In both cases, all features within the volumes of pre-defined distance tolerance are considered contacts. This approach can easily lead to expensive force computation, and it cannot be performed at the desired force update rates.

Instead, we *sample* the contact regions, and compute forces at these points, which are added to yield the global forces to be applied to the objects. These samples are obtained as pairs of "contact" (as defined above) points on the convex pieces of contacting objects. The convex pieces are the result of the convex surface decomposition of the objects and the basic bounding volumes of the BVHs used for intersection detection, as described in section 3. We compute contact points between convex pieces that are either within the distance tolerance or have interpenetrated each other.

In practice, for the disjoint case the *sampled contact points* are the *closest points* between convex pieces, and we obtain them using a collision detection library SWIFT++ [10]. When two objects are inter-penetrating, the sampled contact points are defined by the separating planes between convex pieces, as described in section 3. These contact points form a superset of the local minima of the distance function between both objects around the contact area, so they turn out to be a good approximation for describing the entire contact region.

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

4.2 Clustering Contacts

The surface convex decomposition preprocessing may result in very small convex patches around concavities. This might lead to large number of contacts in certain configurations. The stiffness of different contacts is added up to formulate a single stiffness value, which may cause instability in force feedback.

To avoid instability problems due to large stiffness resulting from such a situation, we group the contacts depending on the Euclidean distance between them. We represent each contact with a single point $\hat{\boldsymbol{p}}$ by taking the average of the contact points in the two objects:

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

Contacts that are closer than a threshold δ from each other are grouped and averaged to a single contact. Hence, given a contact cluster S and a contact c_j:

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

In order to cluster the contacts together, we use octrees and insert them in the appropriate cube-shaped cells. The main diagonal of the cells is set by the distance threshold, δ. This reduces the amount of computation for the clustering operation.

Every cluster is represented by one contact. We need to compute 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.

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

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

where t is the distance tolerance for defining when two disjoint objects are "in contact".

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

$$d = t - \max(t - d_i) \tag{5}$$

4.3 Computation of the Force

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

$$\boldsymbol{F_1}_i = (t - d_i) \cdot \boldsymbol{n}_i \tag{6}$$

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

Next, 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 forces $\boldsymbol{F}_i$:

$$\boldsymbol{T_1}_i = (\hat{\boldsymbol{p}} - \boldsymbol{o_1}) \times \boldsymbol{F_1}_i \tag{8}$$

$$\boldsymbol{T_2}_i = (\hat{\boldsymbol{p}} - \boldsymbol{o_2}) \times \boldsymbol{F_2}_i \tag{9}$$

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 therefore 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 have implemented a force shading algorithm to reduce the discontinuities that happen when the contacts occur across some edge of the models [33]. We use the Gouraud shading algorithm for computing the normal direction at the contact points, 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 type stick-slip presented in [19]. 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.

5 Implementation and Results

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

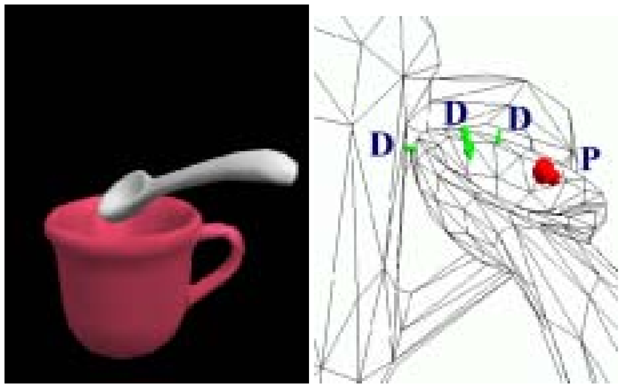
5.1 Experiment Description

We performed our experiments on a Windows 2000 PC with dual 1 GHz Pentium III CPUs and 500 MB memory with a 6DOF 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. Figure 4 illustrates a typical situation in our haptic simulation framework. In this figure, a non-convex object, a spoon, is touching the surface of another non-convex object, a cup. In this particular configuration, the contacts returned by the collision detection module are clustered in four groups. As a result, we have one contact (P) due to the 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.

Model	Convex Pieces	Triangles
torus	67	2000
cup	190	500
spoon	78	336
armadillo	9098	31240
intestine	9913	24846

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



(a) (b)

Figure 4. A Contact Scenario for Cup and Spoon Models. Figure in the right shows the forces computed at clustered contacts, for both disjoint (D, green arrows) and penetrating (P, red arrow) situations.

5.2 Results

Figure 5 illustrates a typical timing profile of a cup interacting with a spoon scenario as shown in Figure 4. 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 interpenetrating (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 of the underlying operating system.

Figures 6-(a), -(b), and -(c) track the computed results on a contact point, the corresponding contact normal, and the distance

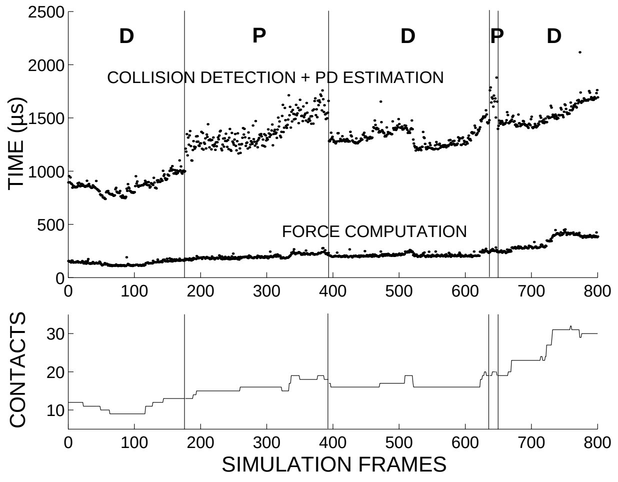


Figure 5. Computation Profile. Recorded while the cup model touches the spoon model. Upper chart: the computation time for the contact query and the force computation. Lower chart: the number of contacts. Vertical lines indicate the intervals of disjoint (D) and interpenetrating (P) contact

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 interpenetrating. In 6-(c), the negative distance values also mean that the objects are interpenetrating. One can notice the smooth transition of the contact information between disjoint and interpenetrating 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.

5.3 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 interpenetrate, the PD computation adds a small increase to the contact query time, as can be noticed from Figure 5.

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. Due to the lack of space, we refer readers to see [23] for extensive experimental results on our PD computation and its implementation, DEEP[3]. Roughly, the PD computation time takes about 0.1 msec using a single 1 GHz Pentium III CPU and Linux operating system, regardless of the complexity of objects.

[3]Dual-space Expansion for Estimating Penetration depth

The complexity of force computation is sub-linear in the number of 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 an intestine 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 [16] penetration depth is roughly estimated along the direction of motion using the previous closest feature pairs, whereas ours 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 sampling problems inherent to volumetric approaches.

6 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. multiresolution 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

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

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

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

[4] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94*, pages 23–34. ACM SIGGRAPH, 1994. ISBN 0-89791-667-0.

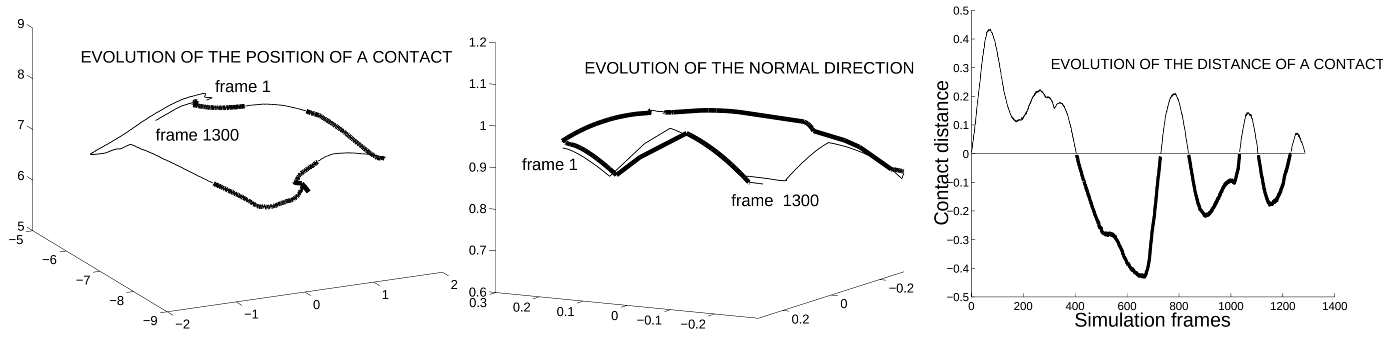
[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. on Management of Data*, pages 322–331, 1990.

[6] G. Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game Developers Conference*, 2001.

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

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

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



(a) (b) (c)

Figure 6. Contact Profile for a Single Contact Scenario. The thick solid line represents that the objects interpenetrate, and the thin solid line indicates that the objects are disjoint. Notice the smoothness of the contact data.

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

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

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

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

[14] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

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

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

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

[18] Leonidas J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

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

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

[21] Philip M. Hubbard. Collision detection for interactive graphics applications. *IEEE Trans. Visualization and Computer Graphics*, 1(3):218–230, September 1995.

[22] D. Johnson and E. Cohen. A framework for efficient minimum distance computation. *IEEE Conference on Robotics and Automation*, pages 3678–3683, 1998.

[23] Y. Kim, M. Lin, and D. Manocha. An incremental algorithm for penetration depth computation between convex polytopes. Technical report, Dept. of Computer Science, UNC-CH, 2001.

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

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

[26] M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

[27] William Mark, Scott Randolph, Mark Finch, James Van Verth, and Russell M. Taylor II. Adding force feedback to graphics systems: Issues and solutions. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 447–452, 1996.

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

[29] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 29–38, August 1990.

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

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

[32] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 289–298, August 1988.

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

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

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

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

[37] Inc. SensAble Technologies. $ghost^{TM}$: Sofware developer's toolkit. *Programmer's Guide*, 1997.

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