Incremental Penetration Depth Estimation Between Convex Polytopes Using Dual-space Expansion

Young J. Kim, Ming C. Lin and Dinesh Manocha

Abstract—We present a fast algorithm to estimate the penetration depth between convex polytopes in 3D. The algorithm incrementally seeks a "locally optimal solution" by walking on the surface of the Minkowski sums. The surface of the Minkowski sums is computed *implicitly* by constructing a local dual mapping on the Gauss map. We also present three heuristic techniques that are used to estimate the initial features used by the walking algorithm. We have implemented the algorithm and compared its performance with earlier approaches. In our experiments, the algorithm is able to estimate the penetration depth in about a milli-second on an 1 GHz Pentium PC. Moreover, its performance is almost independent of model complexity in environments with high coherence between successive instances.

Index Terms—Penetration Depth, Minkowski Sums, Gauss Map, Incremental Algorithm, Haptic Rendering.

1 Introduction

Computing a distance measure between geometric objects is an important problem in robotics, virtual environments and interactive computer games. When two objects are disjoint, the minimum Euclidean distance between them is one of the commonly used distance measure. However, when the objects are overlapping, the Euclidean distance does not give any useful information related to the extent of intersection or penetration. Therefore, a different distance measure is needed to determine the amount of penetration between two overlapping objects [1].

A number of algorithms have been proposed for computing the Euclidean (or separation) distance between two objects. These can be classified into specialized algorithms for convex polytopes or general algorithms for polygonal models based on bounding volume hierarchies. However, many applications like robot motion planning, dynamic simulation or haptic rendering need to know the extent of penetration between overlapping objects. These include robot motion planning in environments consisting of narrow passages [2], six-degree-of-freedom haptic rendering involving object-object interactions [3], [4], contact force computation for dynamic simulation [5], [6].

The natural extension of Euclidean separation distance to overlapping objects is the *penetration depth* (PD) or *intersection depth*. The PD of two inter-penetrating objects A and B is defined as the minimum translation distance that one object undergoes to make the interiors of A and B disjoint. Formally, let P and Q be two intersecting polytopes. Then, the PD of polytopes P and Q, $PD(P,Q)$, is defined as:

$$min\{\| \boldsymbol{d} \| \mid interior(P+\boldsymbol{d}) \cap Q = \emptyset\} \quad (1)$$

One of the commonly used metrics for representing and computing PD's is in terms of Minkowski sums of two objects. The Minkowski sums of $A \oplus B$ are defined as a set of pairwise sums of vectors from A and B; i.e.,

$$A \oplus B = \{\boldsymbol{p}+\boldsymbol{q} |\ \boldsymbol{p} \in A, \boldsymbol{q} \in B\} \quad (2)$$

If we define $-B$ of B as reflecting B with respect to the origin, i.e., $-B = \{-\boldsymbol{q} | \boldsymbol{q} \in B\}$, then the Minkowski sums of $A \oplus -B$, or also known as Configuration Space Obstacle (CSO), are defined as

$$A \oplus -B = \{\boldsymbol{p}-\boldsymbol{q} |\ \boldsymbol{p} \in A, \boldsymbol{q} \in B\} \quad (3)$$

Without loss of generality, let us assume that polytopes A and B are defined with respect to the global origin $\boldsymbol{o}$. Then, if two polytopes P and Q intersect, then the origin $\boldsymbol{o}$ is inside of $P \oplus -Q$, and $PD(P,Q)$ corresponds to the minimum distance from $\boldsymbol{o}$ to the surface of the Minkowski sums of $P \oplus -Q$ [7]. Also notice that if P and Q do not intersect, then $\boldsymbol{o}$ is outside of $P \oplus -Q$, and the Euclidean distance between P and Q corresponds to the minimum distance from $\boldsymbol{o}$ to the surface of $P \oplus -Q$ [8]. Therefore, the unified computational framework based on Minkowski sums provides a continuum of the distance measure between the two objects as they alternate between separation and interpenetration configurations.

The simplest algorithm for PD computation involves computing the Minkowski sums and computing the closest point on its surface from the origin. The worst case complexity of the overall PD algorithm is governed by the complexity of computing Minkowski sums, which can be $O(n^2)$ for convex polytopes and $O(n^6)$ for general (or non-convex) polyhedral models [9]. Since our PD algorithm always considers the Minkowski sum $A \oplus -B$, throughout the rest of the paper, the Minkowski sum refers to the CSO, $A \oplus -B$.

. This research is 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, and Intel

• The authors are with the Department of Computer Science at University of North Carolina at Chapel Hill, U.S.A. Email: {youngkim,lin,dm}@cs.unc.edu

The exact computation of Minkowski sum can be very expensive for interactive applications, like haptic rendering or real-time simulation. Moreover, no robust implementations are known for computing the optimum PD between polyhedral models, especially non-convex polyhedra. As a result, the recent trend has been on computing a good approximation (or estimate) for penetration depth. A number of algorithms have been proposed that differ based on their approximation of the PD [7], [10] or discretization of object's space [11], [12]. However, their implementations are either too slow for real-time applications or they are not very accurate.

1.1 Main Results

We present an incremental algorithm to compute PD for convex polytopes in 3D. Our algorithm uses a number of techniques to initialize some features on the surface of the CSO, which are presumably very close to the features that realize the optimum PD. Then, the algorithm incrementally marches towards a "locally optimal" solution by walking on the surface of the CSO. We define the locally optimal PD using the features on the CSO as follows. Let f be a feature on the CSO that corresponds to 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 on the CSO.

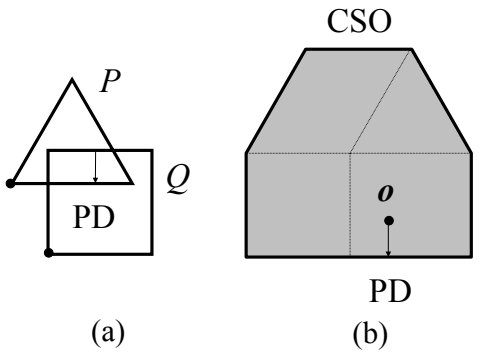


Fig. 1. PD and CSO. CSO is defined as $P \oplus -Q$, and PD corresponds to the minimum distance from the origin to the surface of CSO.

We implicitly compute the surface of the CSO by constructing a local Gauss map and performing a local walk on the polytopes. Our algorithm performs incremental computations and exploits spatial and temporal coherence between successive frames. Our approach for locally computing the Gauss map is based on an earlier algorithm for width computation between convex polytopes.

The resulting algorithm has been implemented and we have tested its performance on a number of benchmarks. In practice, the running time is a fraction of a milli-second on a 1 GHz PC, when there is high motion coherence present in the environment. Furthermore, in most cases, the locally optimal PD that our algorithm computes coincides with the global optimum PD between the underlying polytopes. It also outperforms a more recent algorithm [10] in terms of accuracy and consistency of the result, as well as the runtime performance. For relatively complicated objects, our algorithm runs approximately *six* times faster than the algorithm presented in [10]. A preliminary version of this paper has appeared in [13].

1.2 Organization

The rest of paper is organized as follows. In Section 2, we briefly review the previous work related to penetration depth computation. In Section 3, we describe our incremental algorithm and prove that our algorithm correctly finds locally optimal PD. In Section 4, we present the experimental results from our implementation and compare its performance with earlier algorithms.

2 Previous Work

In this section, we give a brief overview of previous work on collision detection and distance computation, penetration depth and width 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. Check out [14] for a recent survey. 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 [15], incremental computation of Minkowski sums [7], [8], feature tracking based on Voronoi regions [16], [17] and multi-resolution methods [18], [19]. Some of these algorithms are based on incremental computations and exploit frame-to-frame coherence [7], [16], [17].

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 [20], OBB trees [21], sphere trees [22], k-dops [23], Swept Sphere Volumes [24], and convex hull-based trees [25].

2.2 Penetration Depth Computation

Several algorithms have been proposed to compute or estimate the PD. The Minkowski sums of two convex polytope can be computed in $O(n^2)$ worst-case time by computing the overlaying convex planar subdivisions of the Gauss map [26]. Hence, a straightforward algorithm to compute the PD running in $O(n^2)$ time can be immediately devised [1].

Dobkin et al. [9] have presented a hierarchical algorithm that computes the directional PD (i.e. the penetration direction is given) using Dobkin and Kirkpatrick polyhedral hierarchy. For any direction d, it computes the directional penetration depth in $O(\log n \log m)$ time for two convex polytopes with n and m vertices [9]. Agarwal et al. [27] have presented a randomized approach to compute

the PD values. As of now, it is known to be the fastest theoretical algorithm to compute the PD and it runs in $O(m^{\frac{3}{4}+\epsilon}n^{\frac{3}{4}+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon})$ expected time for any positive constant ϵ. However, we are not aware of any implementation of this algorithm.

Given the complexity of optimal penetration depth computation, many approximate algorithms have been proposed for quick estimation. Cameron [7] has presented an extension to the GJK algorithm for separation distance computation [8] to compute upper and lower bounds on the PD between convex polytopes. Van Bergen further elaborated this idea in his expanding polytope algorithm [10]. The algorithm iteratively improves the result of the PD computation by expanding a polyhedral approximation of the Minkowski sums of two polytopes.

The general algorithms for penetration depth estimation are based on discretization of the object space containing the objects. Fisher and Lin [12] have presented a PD estimation algorithm based on the distance field computation and the fast marching level-set method. It is applicable to all polyhedral objects as well as deformable models, and it can also check for self-penetration. Hoff et al. [11], [28] have proposed an approach based on graphics hardware and multi-pass rendering for different proximity queries between general rigid and deformable models, including penetration depth estimation. It uses a combination of object space and image space algorithms to estimate local penetration depth.

Other metrics to characterize the intersection between two objects include the *growth distance* defined by Gilbert and Ong [29]. It unifies the distance measure regardless of whether the objects are disjoint or overlapping and is different from the PD between two inter-penetrating objects.

2.3 Width computation

Given a convex polytope P, its width is defined as the minimum distance between parallel planes supporting P. A simplest width computation algorithm by Houle and Toussaint [30] runs in $O(n \log n + I)$ steps, where n is the number of vertices and I is the number of *antipodal pairs* of edges, and in the worst case $I = n^2$. This algorithm has been implemented by Schwerdt et al. [31] using exact arithmetic. The resulting implementation is quite robust, but somewhat slow and takes about 7.9 seconds for about 1000 points.

Chazelle et al. [32] have used Megiddo's parametric searching technique to improve the width computation. The algorithm runs in $O(n^{\frac{8}{5}+\epsilon})$ time for any $\epsilon > 0$. Agarwal and Sharir [33] formulated width computation as a geometric optimization problem for bichromatic pair of lines for two "vertically-separated" sets of lines. They used a randomized algorithm to compute a closest pair of bichromatic pair of lines, and this result was directly applied to computing the width. The randomized algorithm runs in $O(n^{\frac{3}{2}+\epsilon})$ expected time for any $\epsilon > 0$.

3 Incremental Penetration Depth Computation

In this section, we present our incremental PD computation algorithm for convex polytopes. We also show how the algorithm converges to a locally optimum solution.

3.1 Notation

We use bold-faced letters to distinguish a vector from a scalar value (e.g. the origin, $\boldsymbol{o}$). We assume that the model (a convex polytope in our case) is triangulated and its topological representation is precomputed using such as the quad-edge data structure [34]. Moreover, we use V, E and F, respectively, to denote a vertex, an edge and a face of a polytope. In particular, we use (V,V) to denote the *vertex hub pair* which plays a key role in our incremental algorithm to be explained in Section 3.3. However, we use italic letters to distinguish a particular instance of a feature (e.g. a vertex v in a polytope P) from its generic feature type.

3.2 Width Computation and Penetration Depth

Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in 3D, the *width* of P, $\mathrm{W}(P)$, is defined as the minimum distance between parallel planes supporting P. The width $\mathrm{W}(P)$ of convex polytopes A and B is closely related to the penetration depth $\mathrm{PD}(A, B)$, since it is easy to show that $\mathrm{W}(P) = \mathrm{PD}(P, P)$. Therefore, once a width computation algorithm is readily available, it can be modified to compute the PD. In fact, the asymptotically fastest PD computation algorithm by Agarwal et al. [27] is based on the earlier width computation algorithm proposed by Agarwal and Sharir [33]. A simple algorithm to compute $\mathrm{PD}(A,B)$ based on width computation is formulated as (also see Fig. 2):

1. Find supporting planes P_1 and P_2 on A and B, respectively, whose outward face normals are in opposite directions, and call their inter-distance d_i. For instance, in Fig. 2, there are six d_i's for two intersecting triangles.
2. $\mathrm{PD}(A, B) = \min d_i$, for all i. In Fig. 2-(a), d_5 turns out to be an optimal PD value and based on that value, one can separate the triangles as shown in Fig. 2-(b).

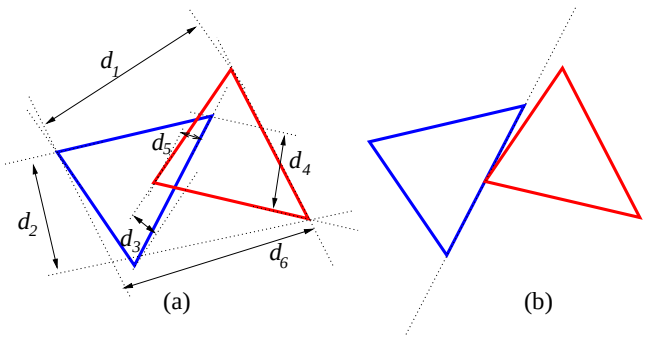
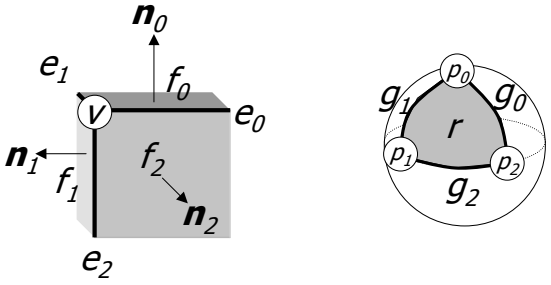


Fig. 2. Brute-force PD computation. Among six possible supporting planes, d_5 is the optimal PD value.

Our incremental PD algorithm is quite similar to Houle and Toussaint's width computation algorithm [30]. Their main idea is based on the following lemma:

LEMMA 3.1 *[30] The width of a set of points P in 3D is the minimum distance between supporting planes, and the plane having the minimum inter-distance (i.e. width) is realized only either by antipodal VF pair or by antipodal EE pair.*

Both Houle and Toussaint's algorithm and our approach only search VF and EE antipodal pairs, and seek a witness feature pair respectively, for the width and the PD value. Hence, the main issue in both algorithms becomes finding such VF and EE antipodal pairs. Houle and Toussaint's width algorithm accomplishes it by using the standard dual mapping on the Gauss map (or normal diagram). The mapping is defined from object space to the surface of a unit sphere $\mathbb{S}^2$ in 3D [35]. 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. Then, the algorithm finds the antipodal pairs by overlaying the upper hemisphere of the Gauss map on the lower hemisphere and computing the intersections between them.



(a) Features (b) Gauss Map

Fig. 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 g_0, g_1, g_2, and the vertex v to a convex region r.

3.3 Algorithm Overview

In our incremental PD computation algorithm, we do not compute the entire Gauss map for each polytope or their entire Minkowski sums. Rather we compute them in a lazy and incremental manner, based on local optimization. Starting from some feature on the surface of the Minkowski sums, the algorithm finds the direction in which it can decrease the PD value and proceeds towards that direction by extending the surface of the Minkowski sums.

At each iteration of the algorithm a vertex is chosen from each polytope to form a pair. We label it as a *vertex hub pair* and use it as a hub of the expansion of the local Minkowski sums. The vertex hub pair is chosen in such a way that there exists a plane supporting each polytope, and it is incident on each vertex. It turns out that the vertex hub pair corresponds to two intersected convex regions on the Gauss map, which later become intersecting convex polygons on the plane after *central projection*. The intersection of convex polygons correspond to the VF or EE antipodal pairs from which one can reconstruct the local surface of the Minkowski sums around the vertex hub pair. Given these pairs, we choose the one that corresponds to the shortest distance from the origin of the Minkowski sums to their surface. If this pair decreases the estimated PD value, we update the current vertex hub pair to an appropriate one which is adjacent to the chosen antipodal pair. We iterate this procedure until we can not decrease the current PD value and converge to a local minima. The details of the algorithm are given below.

3.4 Initialization

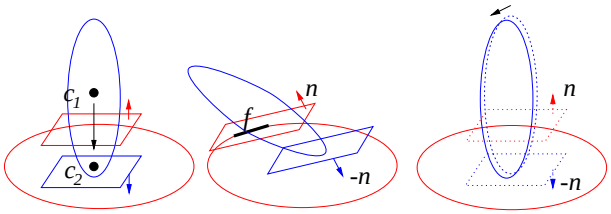
The algorithm starts with an initial guess on the vertex hub pair, (V,V), on the polytopes (a region/region pair on the Gauss map). The initial guess is important for the performance of our incremental algorithm. A good initial guess can lead to empirically "almost constant" running time, whereas a bad 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 and some of them are application dependent. The goal is to estimate the optimal penetration direction. Once the estimated direction is computed, we take the extremal vertex of each polytope along that direction and use it to form the vertex hub pair. We present three heuristic techniques to estimate the optimal penetration direction. We demonstrate in our extensive experiments in Section 4.2 that these simple techniques work quite well in practice.

A good estimate to the penetration direction can be obtained by taking the centroid difference between objects, and computing an extremal vertex pair for the difference direction. For instance, in Fig. 4-(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 work well for initial guess on closest features for Voronoi Marching based separation distance computation algorithm [18], and also works well for estimating the penetration direction.

In other cases, the penetrating features can also suggest a good initial guess. When the objects penetrate, many of the proximity query algorithms report a witness feature pair for it [16], [18]. From this feature pair, one might be able to estimate the direction or compute the actual feature pair that corresponds to the optimal PD. One possible way is to consider the plane normal of a penetration feature as penetration direction. For instance, in Fig. 4-(b), the face

f is identified as a penetration witness, and its associated plane normal n is used for the extremal vertex query.

Many applications exhibit high spatial or temporal coherence between successive frames. In such environments with high motion coherence, the PD computation result from the previous time frame or the closest features between non-overlapping objects can provide a good guess for the next time frame. In Fig. 4-(c), for example, the previous PD features[1] provide a direction for the extremal vertex query.



(a) Centroid Difference (b) Penetration Features (c) Motion Coherence

Fig. 4. 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.

3.5 Iterative Optimization

After the algorithm obtains a initial guess for a (V,V) pair, it iteratively seeks to improve the PD estimate 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 walking to a pair which provides a greatest improvement in the PD value. In more detail, let the current vertex hub pair be (v_1, v_1'). The next vertex hub pair (v_2, v_2') is computed as follows (also its pseudo code is described in Algorithm 3.1 and 3.2):

1. Construct a local Gauss map each for v_1 and v_1' (step 1 of Algorithm 3.2).
2. Project the Gauss maps onto $z = 1$ plane, and call them G and G' respectively. G and G' correspond to convex polygons in 2D (step 1 of Algorithm 3.2).
3. Compute the intersection between G and G' using a linear time algorithm such as [36]. The result is a convex polygon and we label each vertex comprising the intersection u_i. These u_i's correspond to the VF or EE antipodal pairs in object space (step 2 of Algorithm 3.2).
4. In object space, determine which u_i corresponds to the best local improvement in PD (step 3 of Algorithm 3.2).
5. Set an adjacent vertex pair (adjacent to u_i) to (v_2, v_2') (Algorithm 3.1).

This iteration is repeated until either there is no more improvement in the PD value or number of iterations reach some maximum value.

At step 5 of the iteration, the next vertex hub pair is selected in the following manner. If u_i corresponds to VF, then we must choose one of the two vertices adjacent to F assuming that the model is triangulated. The same reasoning is also applied to when u_i corresponds to EE. As a result, we need one more iteration in order to actually decide which vertex hub pair we want to select. However, we cache the results of this extra iteration and use it for future computations.

3.6 Correctness of the Algorithm

In this section, we will show that the incremental algorithm always terminates and computes a locally optimal PD. The termination of the algorithm is guaranteed as the PD value computed by the algorithm is decreasing. Moreover the algorithm computes a local minimum, as the incremental walk is performed on the surface of the CSO. A snapshot of a typical step during the iteration is illustrated in Fig. 5.

Find_Incremental_PD(vertex $\boldsymbol{v_1}$, vertex $\boldsymbol{v_1'}$)
Input A starting (V,V) pair, $\boldsymbol{v_1 v_1'}$.
Output A locally optimal PD value.
$\{ T_1, d_1 \}$ = **Report_Min_Pair**($\boldsymbol{v_1}$, $\boldsymbol{v_1'}$)
Repeat {
  switch(T_i){
    Case EE ($T_i = \{e_i, e_i'\}$):
      v_{i_1} is an incident vertex of e_i other than v_i.
      v_{i_1}' is an incident vertex of e_i' other than v_i'.
      $\{T^1, d^1\}$ = **Report_Min_Pair**($\boldsymbol{v_i}$, $\boldsymbol{v_{i_1}'}$).
      $\{T^2, d^2\}$ = **Report_Min_Pair**($\boldsymbol{v_{i_1}}$, $\boldsymbol{v_i'}$).
      if ($d^1 < d^2$)
        $\{\boldsymbol{v_{i+1}}, \boldsymbol{v_{i+1}'}\} = \{\boldsymbol{v_i}, \boldsymbol{v_{i_1}'}\}, T_{i+1} = T^1, d_{i+1} = d^1$
      else
        $\{\boldsymbol{v_{i+1}}, \boldsymbol{v_{i+1}'}\} = \{\boldsymbol{v_{i_1}}, \boldsymbol{v_i'}\}, T_{i+1} = T^2, d_{i+1} = d^2$
    Case VF ($T_i = \{v_i, f_i'\}$):
      v_{i_1}' and v_{i_2}' are incident vertices of f_i' other than v_i'.
      $\{T^1, d^1\}$ = **Report_Min_Pair** ($\boldsymbol{v_i}$, $\boldsymbol{v_{i_1}'}$).
      $\{T^2, d^2\}$ = **Report_Min_Pair** ($\boldsymbol{v_i}$, $\boldsymbol{v_{i_2}'}$).
      if ($d^1 < d^2$)
        $\{\boldsymbol{v_{i+1}}, \boldsymbol{v_{i+1}'}\} = \{\boldsymbol{v_i}, \boldsymbol{v_{i_1}'}\}, T_{i+1} = T^1, d_{i+1} = d^1$
      else
        $\{\boldsymbol{v_{i+1}}, \boldsymbol{v_{i+1}'}\} = \{\boldsymbol{v_i}, \boldsymbol{v_{i_2}'}\}, T_{i+1} = T^2, d_{i+1} = d^2$
    Case FV:
      It is similar to the VF case.
  }
} **until** (d_i is non-decreasing)
return d_i.

ALGORITHM 3.1: Find_Incremental_PD

LEMMA 3.2 *The optimal PD of two convex polyhedra is the minimum distance between supporting planes through either a VF or EE antipodal pair.*

Proof: The proof is very similar to that of Lemma 3.1.

1. By the PD feature, we mean a pair of features on both polytopes whose supporting planes realize the locally optimal PD value.

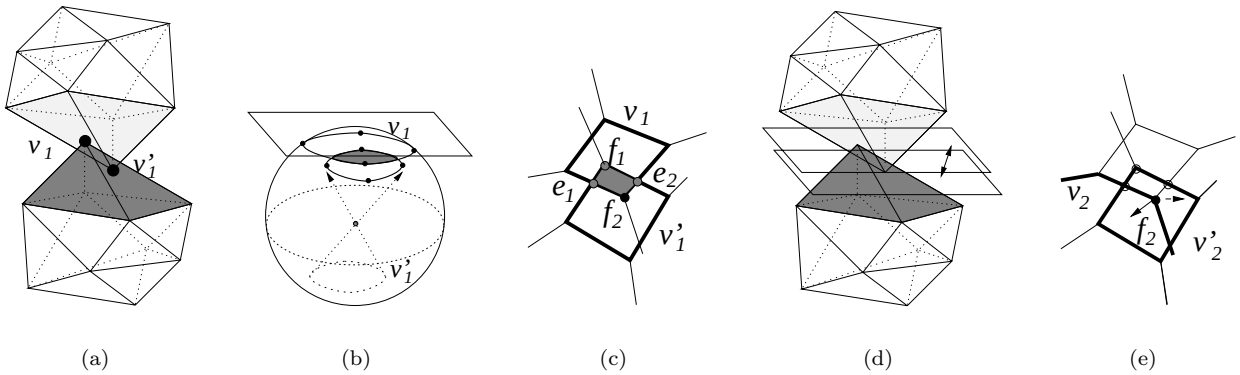


Fig. 5. Iterative Optimization: (a) The current (V,V) pair is $\boldsymbol{v_1 v_1'}$ and a shaded region represents edges and faces incident to $\boldsymbol{v_1 v_1'}$. (b) shows local Gauss maps and their overlay for $\boldsymbol{v_1 v_1'}$. (c) shows the result of the overlay after central projection 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 $\boldsymbol{v_2 v_2'}$ is determined as the next vertex hub pair.

Report_Min_Pair(vertex $\boldsymbol{v_i}$, vertex $\boldsymbol{v_i'}$)

Input A vertex hub pair, $\boldsymbol{v_i v_i'}$.

Output A feature pair, T, which has a greatest improvement on the PD value, and the PD value, d.

1. Construct a Gauss map for v_i and v_i', and perform the central projection.
2. Using the convex polygon intersection algorithm, compute all EE intersections along with VF or FV inclusion in $O($ number of incident edges in v_i and v_i' $)$ time.
3. For every EE, VF, FV pair, find a pair which generates a minimum inter-distance between supporting planes, and call the pair, T.
4. Return the minimum inter-distance d along with T.

ALGORITHM 3.2: Report_Min_Pair

LEMMA 3.3 *Given two convex polytopes, P and Q, and points $\boldsymbol{p} \in P$ and $\boldsymbol{q} \in Q$, $\boldsymbol{p} - \boldsymbol{q} \in \partial(P \oplus -Q)$ if and only if there exist supporting planes for $\boldsymbol{p}$ and $\boldsymbol{q}$ with opposite directions of the normals. The ∂ denotes the boundary and $\oplus$ denotes the Minkowski sums.*

Proof: See [37] for the proof.

THEOREM 3.1 *The given incremental algorithm for PD estimation computes a locally optimal PD value.*

Proof: By Lemma 3.2, the algorithm considers (only) necessary antipodal pairs adjacent to the initial VV (hub) pair in order to reduce the PD value. By Lemma 3.3, the algorithm walks on the boundary of Minkowski sum $P \oplus -Q$, which is a convex polytope. Therefore, the algorithm computes a locally optimal PD value. Q.E.D.

3.7 Analysis of PD Algorithm

In the worst case, our PD algorithm can take $O(n^2)$, where n is the number of faces in polytopes, since the algorithm might need to explore all the faces of the Minkowski sums. However, in our experiment (as explained in Section 4.2), we have found out that the algorithm is terminated after less than five iterations. Moreover, the expected operation count of the algorithm is about $186k + 154$ per iteration including the cost of transformation, where k is the number of faces adjacent to a vertex hub pair and typically k is a small number in practice. Among them, $166k + 6$ operations are part of the polygon intersection routine, and $20k + 115$ are taken by the Gauss map construction. Therefore, the algorithm shows constant time performance in practice. Regarding the space requirement of our algorithm, each iteration needs $O(k)$ temporary storages to construct and maintain the local Gauss maps.

3.8 Local vs Global Minimization

Since our incremental algorithm is a local minimization process, it can get stuck in a local minimum. This can happen based on the choice of the initial pair of features.

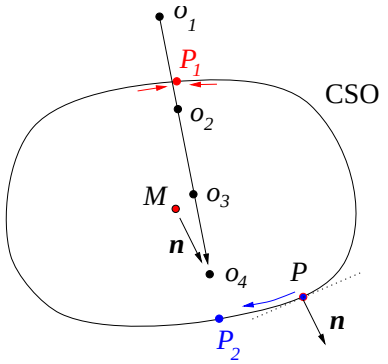


Fig. 6. Escaping from a local minimum: As the origin of the CSO moves from $\boldsymbol{o_1}$ to $\boldsymbol{o_4}$, the PD algorithm always reports P_1 as a PD feature. However, using the "centroid difference" vector n, the algorithm starts the search from P, and marches toward P_2.

For example, in Fig. 6, as the origin of the CSO moves from $\boldsymbol{o_1}$ to $\boldsymbol{o_4}$, the PD algorithm uses a PD feature from previous computation as a starting point of the walk. As a

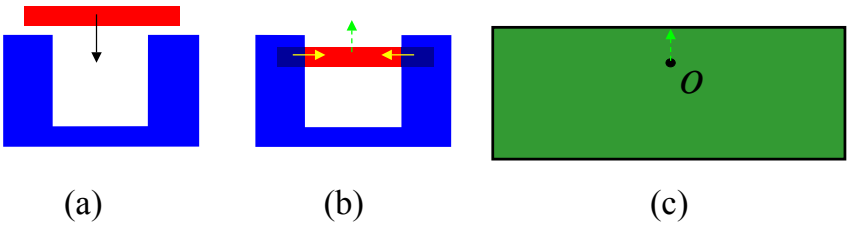


Fig. 7. Local vs Global PD Computation. (a) shows the situation before two polygons in 2D come into contact. (b) shows $O(nm)$ intersections after the polygons are intersected. However, a localized PD computation (denoted by solid yellow arrows) based on $O(nm)$ intersections may not provide a global PD which is denoted as a dotted green arrow in this figure. (c) shows the Minkowski sum of the two polygons in (b). The minimum distance from the origin to the surface of the Minkowski sum corresponds to the global PD.

result, the algorithm reports P_1 as a locally optimum PD feature at each frame. In this case, the initial estimate at $\boldsymbol{o_4}$, P_1, is not a good choice as the true optimal PD feature is P_2. In practice, we can avoid such cases by employing different heuristics suggested in Section 3.4. In this particular example, the "centroid difference" heuristic solves the problem. In the Fig. 6, M is the center of mass of the CSO, and $\boldsymbol{n}$ is the centroid difference vector when the origin of the CSO is at $\boldsymbol{o_4}$. Then, if the algorithm is provided with an additional guess at P, it finds that P is a better estimate as compared to P_1 and starts towards P_2.

A more generic way to escape from the local minimum problem is to employ a global search mechanism. A simple way is to use a discretization approach. As suggested in [27], one might sample the search space (e.g. the Gaussian space in our case), and for each sampled search direction compute the minimum. Obviously this approach can be expensive depending on the size of sample space. Another way of the global search is to approximate the whole CSO and launch a search on the approximation. Van Bergen presented one such algorithm [10]. The major drawback of this approximation scheme is its numerical instability. The numerical errors add up with each iteration and the PD computation based on the incorrect approximation can report inconsistent results. This phenomenon is demonstrated in Fig. 14-(b) $\sim$ Fig 16-(b) in Section 4.

3.9 Extension to Non-Convex Polyhedra

In general, it is difficult to directly extend a PD algorithm for convex polytopes to one for non-convex polyhedra, since the computation of the PD between two general polyhedral models is a global problem and requires Minkowski sums for non-convex polyhedra. Therefore, a local solution computed using some 'divide-and-conquer' approach may not be correct, as illustrated in Fig. 7.

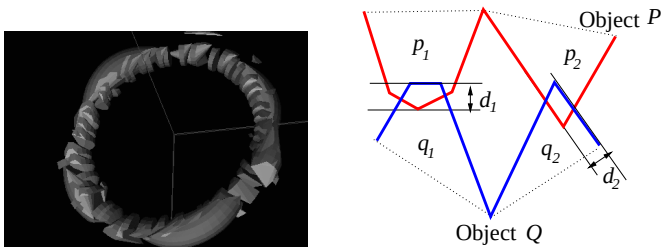
In some situations, instead of reporting one global PD, it is sufficient or sometimes necessary to report a set of PD's for each overlapping convex portion between non-convex polyhedra. These applications include randomized path planning, six-degree-of-freedom (6DOF) haptic rendering and dynamic simulation. In particular, we have successfully demonstrated how to apply this method to 6DOF haptic rendering application [38], and we explain it in detail in Section 4.4. For these applications, even though the pairwise PD computations do not satisfy the definition of global PD as given by Equation 1, there are two benefits from using a set of localized PD's.

First of all, the PD amount treated in the application is relatively small such that the chances are high that the global PD can coincide with one of the localized PD's. This phenomenon is due to the fact that the relative displacement of an object between each motion frame is very small and the penetration situation is usually followed by its resolution to the separation situation. Secondly, a set of PD's can compensate for the lack of consideration for rotational motion in the definition of PD. The rotational motion can be a problem for both convex and non-convex objects, however the problem is more severe for a non-convex object, since it can possibly contain many local minima on its surface. 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 that objects can be separated.

3.9.1 *Localized PD Computation based on Convex Surface Decomposition*

In order to compute localized PD between overlapping convex polytopes, we first decompose each polyhedron into a collection of convex pieces. We use a convex surface decomposition technique for decomposing polyhedra as follows. We decompose the surface of each non-convex polyhedron into a collection of convex surface patches using a greedy walk on the surface. Convex pieces are then formed by taking the convex hull of each surface patch. For example, Fig. 8-(a) shows an example of a convex decomposed torus model. We refer the reader to [25], [39] for detailed discussion on the surface decomposition technique.

Once we have computed the convex decomposition, we compute PD for each overlapping convex pieces between decomposed polyhedra. For instance, in Fig. 8-(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.



(a) A decomposed torus (b) Pairwise estimation

Fig. 8. Extension to Non-Convex Objects

4 Implementation and Performance

In this section, we describe our implementation and highlight the performance of our PD algorithm on different models and environments. Moreover, we compare its performance with a globally optimal PD computation algorithm and another estimation algorithm presented by [10].

4.1 Implementation Issues

There are a couple of issues in implementing our PD algorithm presented above. The first one is related to the collision detection algorithm or library used by our PD algorithm. The second issue is related to handling degeneracies.

4.1.1 Collision Detection Algorithm

A collision detection algorithm is used to check whether two objects overlap. Our prototype implementation, DEEP[2], is tightly coupled with SWIFT collision detection library [18], and it takes advantage of the collision query provided by SWIFT. There are three benefits that we gain from using SWIFT.

First of all, SWIFT makes use of motion coherence between successive frames. Secondly, it tracks closest features between the polytopes and reports a pair of collision witness features that the PD algorithm can utilize as the starting feature pair for the local optimization. Finally, DEEP borrows the application programming interface (API) from SWIFT. Thus, to the user, the PD query does not look any different than the separation query, except that the distance result being returned is negative. This unified API provides the user with a convenient and consistent way of handling the distance query in an application.

4.1.2 Handling Degeneracies

Geometric algorithms are prone to degeneracies. In the case for DEEP, there are two major sources of degeneracy that arise in terms of implementation and application. The first one relates to co-planar faces in any of the polytopes, and the other one arises from the central projection used during the construction of the Gauss map.

Promoting Co-Planar Faces. In our framework, co-planar faces should be considered as a degenerate input, because they are mapped to the same point on Gauss map. This causes an immediate problem in the convex/convex polygon intersection routine in Algorithm 3.2, because the intersection algorithm described in O'Rourke et al. [36] is provided with an edge with zero-length and the algorithm can not handle such a case properly. Fortunately, this is relatively easy to solve by simply ignoring those zero-length edges. However, when it comes to finding the next feature pair in Algorithm 3.1, those faces can not be simply ignored. It is possible that one of the co-planar faces may be the current optimal PD witness pair and any vertex hub pair adjacent to the co-planar face can be the next candidate. For example, see Fig. 9.

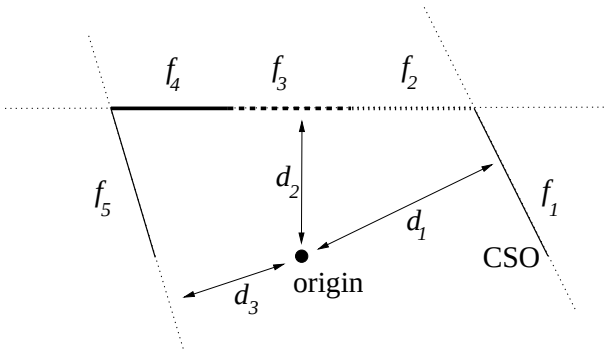


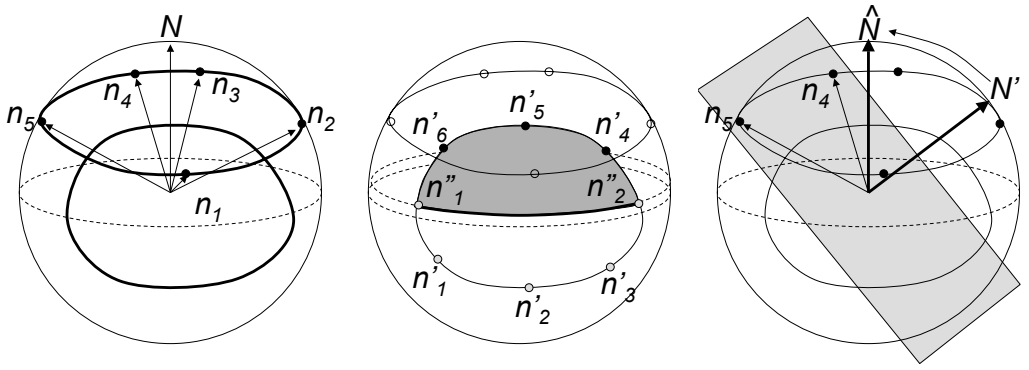
Fig. 9. Degeneracy 1: Co-Planar Faces. Let f_2, f_3, f_4 be the features on a CSO generated by some co-planar faces such that they have the same PD value, d_2. Also assume that f_1 is the current PD feature and d_1 is its associated PD value. Thus f_5 is the locally optimal PD feature and d_3 is the locally optimal PD value. Then, after one iteration, the iterative algorithm finds f_2 as the next PD feature, since its associated PD value d_2 is less than d_1. However, at this point, if the algorithm does not consider other co-planar features such as f_3 and f_4, it stops the iteration since it can not improve the PD value.

Essentially these co-planar features should not be generated if we could relax the underlying modeling constraints (e.g. triangulation). Thus, theoretically, there is nothing wrong with promoting the co-planar faces to their adjacent faces. However, as a result of the promotion the size of neighborhood search grows with the number of co-planar faces.

Local Gauss Map Construction. The central projection during the construction of the Gauss map also results in degeneracies. The central projection maps the equator of the Gauss map to infinity and it also splits the edge crossing the equator. We avoid such cases by constructing the Gauss map locally in the following manner.

For a given vertex hub pair (v_1, v_2), take one vertex region, say v_1. Let us denote the set of normals that contribute to v_1, as n_i's. Our goal is to find a hemisphere H to enclose all the n_i's, and the north pole of H will be the direction of the central projection. At the same time, we

2. DEEP is available for download at http://gamma.cs.unc.edu/DEEP.



(a) (b) (c)

Fig. 10. Degeneracy 2: Gauss Map Construction. (a) Given a set of normals, $\{n_1, n_2, \ldots, n_5\}$, comprising the Gauss map for v_1, the goal is to find N such that $N \cdot n_i \geq 0$. (b) The normals comprising the Gauss map for v_2, $\{n_1'', n_2'', n_4', n_5', n_6'\}$, are obtained by cutting $\{n_1', n_2', \ldots, n_6'\}$ by the hemisphere H of the Gauss map of v_1. The shaded region is the intersected region between the new Gauss map of v_2 and that of v_1. (c) N is computed by incrementally moving $N' = n_k \times n_{k+1}$ toward $\hat{N} = \text{avg}\{n_1, n_2, \ldots, n_5\}$.

also need to minimize the dispersion of n_i's from the north pole of H. The latter is required to reduce the numerical errors that will be induced by the central projection. Otherwise, these errors will significantly affect the convex/convex polygon intersection algorithm in Algorithm 3.2.

For instance, in Fig. 10-(a), $\{n_1, n_2, \ldots, n_5\}$ are the normals comprising the Gauss map of v_1. We want to compute a hemisphere H with its north pole located at N that minimally encloses $\{n_1, n_2, \ldots, n_5\}$. Note that, at this point, it is still possible that the other vertex region v_2 may not be totally enclosed by H such that v_2 still can have a infinite boundary after the central projection, see Fig. 10-(b). We avoid this situation by cutting the Gauss map of v_2 by the equator of H. In fact, we slightly perturb H toward the closest n_i of the Gauss map to H, in order to avoid the boundary at infinity. This does not affect the result of convex/convex intersection routine, since the intersection only exists on the hemisphere H.

For example, in Fig. 10-(b), originally the normals that contain the Gauss map of v_2 are $\{n_1', n_2', \ldots, n_6'\}$. After the cutting operation by the equator of H, the Gauss map now becomes $\{n_1'', n_2'', n_4', n_5', n_6'\}$. The dark region in Fig. 10-(b) denotes the intersection area between the Gauss maps of v_1 and v_2. Here, the cutting operation does not affect the result of the intersection. This cutting operation is performed at run-time.

Back to the problem of finding H, we can formulate this problem as follows. Given a set of normals $\{n_i\}$, we want to find a normal N such that $n_i \cdot N \geq 0$ for all i. Since we also want to minimize the angle between N and n_i (in order to avoid a thin polygon when projected onto a plane), we reduce the problem to a linear programming (LP) problem. To rephrase this, given a set of normals $\{n_i\}$,

$$\text{maximize} \quad \sum N \cdot n_i \quad \text{subject to} \quad N \cdot n_i \geq 0 \quad \text{for all } i. \tag{4}$$

In equation 4, we do not necessarily want the maximal N, as long as N is reasonably close to the maximum and $N \cdot n_i \geq 0$. Furthermore, the above LP problem has an infinite number of solutions. Therefore instead of solving the LP problem, we use the following iterative method.

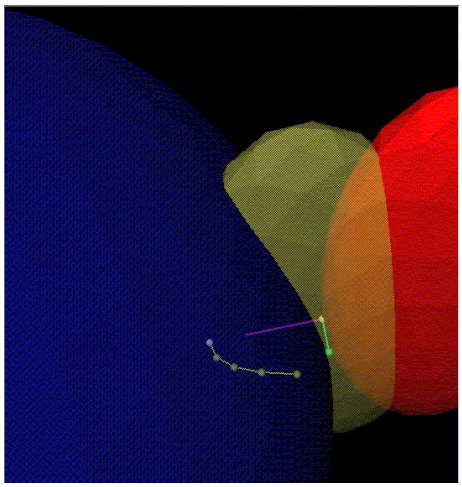
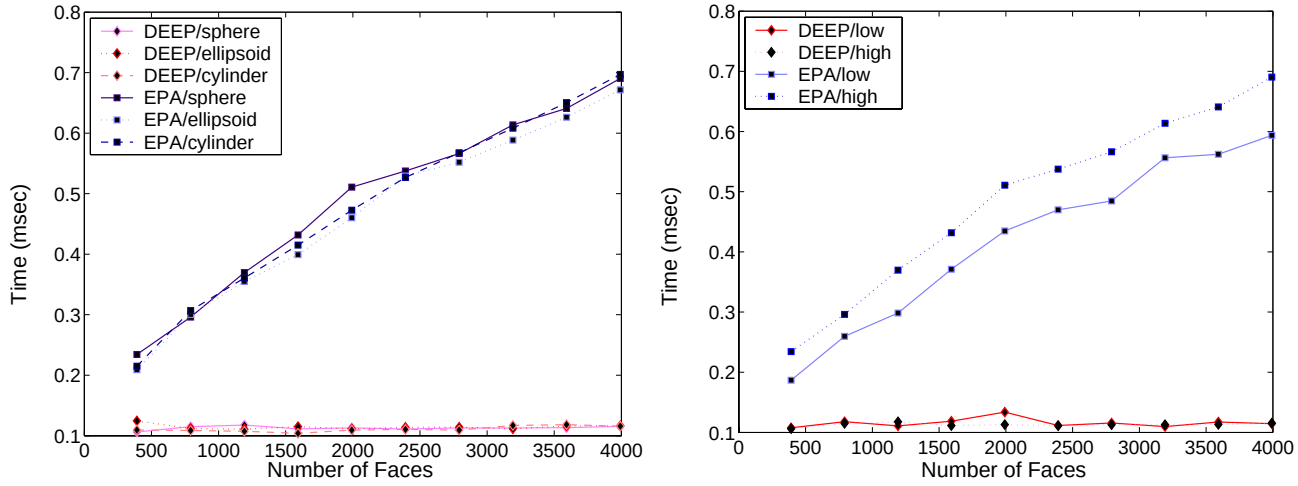


Fig. 11. A typical benchmarking setup: A sphere vs an ellipsoid. The yellow ellipsoid penetrates into the blue sphere. As a result of PD computation, the yellow ellipsoid is translated to the red ellipsoid with the amount of the PD value so that the surface of the red ellipsoid touches that of the blue sphere. Yellow and green solid lines on the sphere and ellipsoid show the history of walking for a vertex hub pair.

Notice in Equation 4 that, without the constraints of $N \cdot n_i \geq 0$, a simple algebra would suggest that there is a single solution for N, which is the average of n_i's. Let



(a) Fixed PD Amount

(b) Variable PD Amount

Fig. 12. Performance Results of Incremental Algorithm: Figure (a) shows the results of DEEP and EPA for the sphere, ellipsoid and pen model with a fixed PD amount. Figure (b) shows the results of DEEP and EPA for the sphere model by changing the PD value. In both figures, upper lines are the results by EPA, and lower lines are by DEEP. We also selected benchmarks with high motion coherence. Note that DEEP is invariant of the complexity of a model and the PD amount.

us call the average $\hat{N}$. We want N as much as close to $\hat{N}$. Since $\{n_i\}$ can be enclosed by a hemisphere H and they form a convex subdivision, any $N' = n_k \times n_{k+1}$ satisfies $N' \cdot n_i \geq 0$ for all i, where $n_k, n_{k+1} \in \{n_i\}$. First, we consider such N' as an initial north pole of the hemisphere H. This would make n_k and n_{k+1} located along the equator of H, which would be projected to infinity. Then, we iteratively rotate N' from the north pole toward $\hat{N}$ while satisfying $N \cdot n_i \geq 0$ for all i. We iterate this process until $N' \cdot n_i \geq 0$ is violated, as illustrated in Fig. 10-(c). We precompute these H's for each vertex, and later at run-time we use any H as a vertex hub pair.

4.2 Performance

All the timings presented in this section were obtained on a Linux PC with 1 GHz Pentium III CPU, 256 MB memory, and g++ compiler. We used different models with varying combinatorial complexity, as well as aspect ratios to test the performance. The model characteristics used in our benchmarking include:

- The number of faces in the testing models vary from 400 to 4000.
- Three basic primitives, a sphere, an ellipsoid, and a cylinder, are used to generate random models of different aspect ratios. Based on a given primitive, random points on its surface were sampled and a polytope was generated by taking their convex hull. A penetrating object was selected out of the three basic types of objects, whereas only a sphere was chosen as the other object. For example, Fig. 11 shows a typical example of our benchmark setup.
- During each time step of the simulation, a penetrating object revolves around a penetrated object, while rotating around its center of mass.

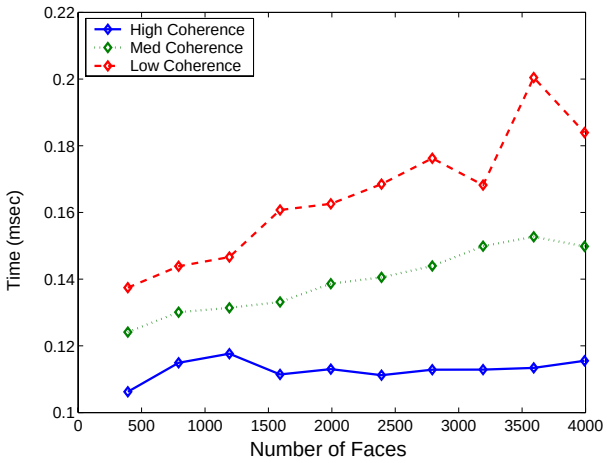
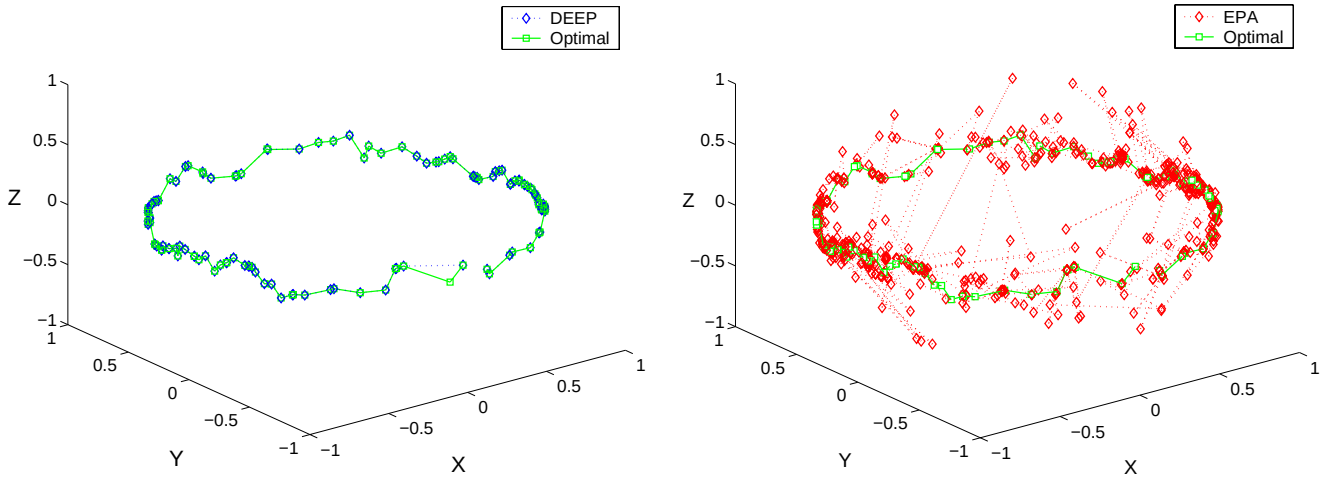


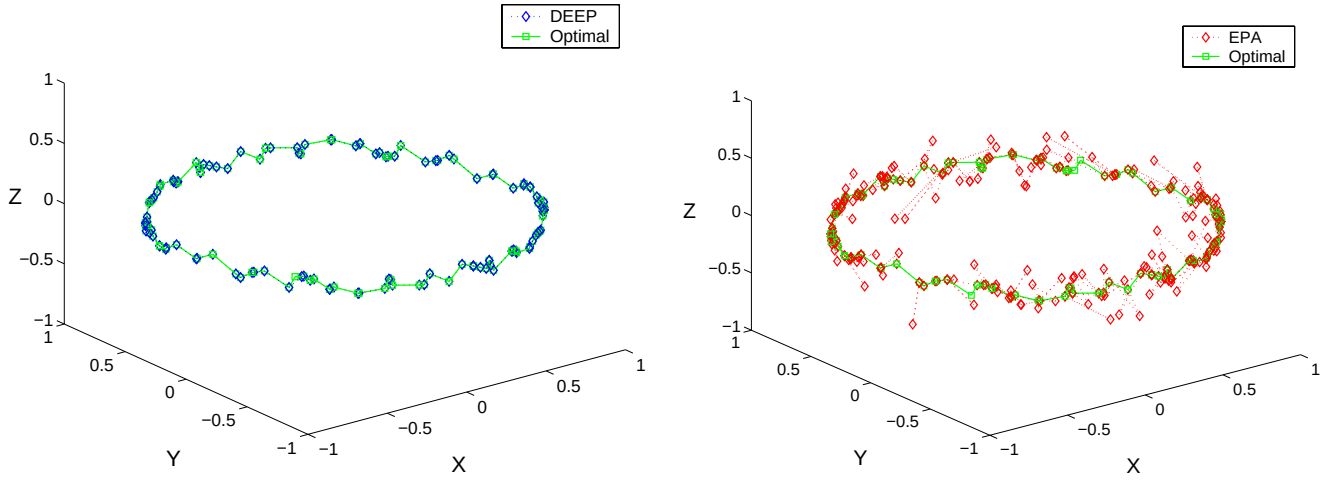
Fig. 13. Performance of DEEP under Different Levels of Motion Coherence: The PD value is fixed to low.

Fig. 12 shows the runtime performance of our incremental algorithm (DEEP), and also compares it with the result of the expanding polytope algorithm (EPA) [10]. We selected different benchmarks with high motion coherence. This figure highlights the empirically observed *almost constant time* performance of DEEP on these benchmarks, especially when there is a high motion coherence present between successive frames in the simulation environment. We classified the level of motion coherence into high, medium, and low. In each time step of simulation, an object moves by the amount of 1/720 (high), 1/144 (medium), 1/72



(a) DEEP (b) EPA

Fig. 14. Tracking Results of Penetration Direction Vectors [Cylinder vs Sphere]. Fig. (a) shows the tracking results of penetration direction vectors from DEEP, and (b) shows that from EPA. In each figure, optimal penetration direction vectors are also annotated for the sake of comparison. The levels of motion coherence and PD value are fixed to high and low, respectively. Note that DEEP rarely misses the optimal direction, but EPA displays quite unpredictable behavior.



(a) DEEP (b) EPA

Fig. 15. Tracking Results of Penetration Direction Vectors (Continued) [Sphere vs Sphere]

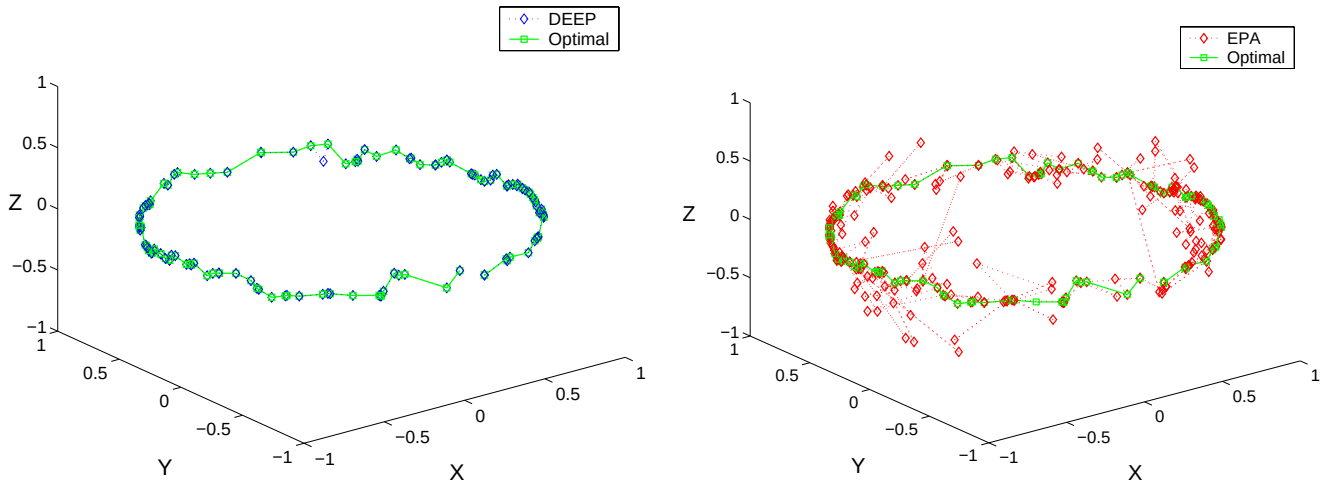
(low) times the size[3] of the object. We have also classified the depth of penetration into high (the PD value is half the size of the object) and low (1/5 times the size of the object). As Fig. 12-(b) suggests, the performance is not affected by the depth of penetration. However, the running time of GJK-based expanding polytope algorithm [10] grows linearly with the number of faces. This is because of global search in the underlying algorithm and it is also a function of the amount of penetration.

3. In our implementation, it is defined as the radius of the minimum bounding sphere.

We also evaluated the performance on different scenarios under varying motion coherence, see Fig. 13. As the motion coherence is reduced, the performance degrades almost linearly. In the environment with low motion coherence, the running time is no longer constant.

4.3 Comparisons with an optimal solution

We compared the results of DEEP with an implementation that computes the globally optimal solution. In this case, the rotational motion was not considered. The globally optimal solution is obtained by explicitly computing the Minkowski sums of two convex polytopes. A simple



(a) DEEP (b) EPA

Fig. 16. Tracking Results of Penetration Direction Vectors (Continued) [Ellipsoid vs Sphere]

brute-force algorithm to compute the Minkowski sum is based on computing the pairwise Minkowski sum for every vertex pair from each polytope, followed by convex hull of resulting $O(n^2)$ combinations.

For convex polytopes, a penetration depth value simply depends on a penetration direction vector. Fig. 14 $\sim$ 16 show tracking results on the penetration direction vectors during the course of the simulation. As Fig. 14-(a) $\sim$ 16-(a) suggest, DEEP rarely misses the optimal penetration direction in our experiments. Thus, DEEP is able to compute the global optimum solution in most cases. In terms of comparison to the brute-force implementation, the mean of the error in the PD value of our algorithm was 3.4711^{-6} and its standard deviation was 7.345^{-5}.

However, in Fig. 14-(b) $\sim$ 16-(b), the expanding polytope algorithm (EPA) shows unpredictable behavior. There are two reasons for this. First of all, the EPA is susceptible to numerical errors, because it can generate many oblong triangles during its polyhedral expansion of Minkowski sums. Secondly, because the EPA is a global search algorithm and its termination condition considers only a PD value, it can stop any time during the expansion whenever the termination condition is satisfied. Therefore, even though the optimal penetration direction vector does not change much between successive frames, the result computed by EPA can change considerably.

4.4 Application to Haptic Rendering

In this section, we highlight the application of our PD computation algorithm to six degree-of-freedom (6DOF) haptic rendering. One such device used for force display is shown in Fig. 17. One of the common techniques used to render the responsive haptic forces is based on *penalty-based methods*. Here, the "penalty" is determined by the amount of interpenetration between overlapping objects.

Fig. 17. 6-DOF Haptic Device: The PHANTOM of SensAble Technologies

Thus, our PD algorithm can be directly applied to the penalty-based 6DOF haptic rendering [38]. Moreover, our PD algorithm is particularly well suited for the haptic rendering application, since the application requires high update rates (e.g. 1 KHz), thus exhibits high motion coherence. We can also apply our PD algorithm to the 6DOF haptic rendering of non-convex models by using the localized PD computation approach explained in Section 3.9.1.

5 Conclusions

We present an incremental PD algorithm to compute a locally optimal PD. It runs in almost constant time when

there is high motion coherency in the environment, and it finds an optimal solution most of times. The algorithm is based on the implicit, local construction of the Minkowski sums and the iterative optimization on the PD value.

There are some pending research issues in the PD computation. One immediate problem is to devise a more generic algorithm that always handles the possible local minimum problem. We are considering the use of multiresolution techniques or randomized algorithms. Computing the PD between non-convex objects is a very challenging problem. Therefore, any estimation technique or solving the PD for a limited class of non-convex objects (e.g. a convex object vs. a non-convex object) can still be highly desirable. Recently, an approach using a combination of object-space and image-space algorithms for general polyhedral models has been presented in [40].

References

[1] S. A. Cameron and R. K. Culley, "Determining the minimum translational distance between two convex polyhedra," in *Proc. of IEEE Inter. Conf. on Robotics and Automation*, 1986, pp. 591–596.

[2] D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," *Proc. of 3rd Workshop on Algorithmic Foundations of Robotics*, 1998.

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

[4] 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.

[5] Michael McKenna and David Zeltzer, "Dynamic simulation of autonomous legged locomotion," in *Computer Graphics (SIGGRAPH '90 Proceedings)*, Forest Baskett, Ed., Aug. 1990, vol. 24, pp. 29–38.

[6] Matthew Moore and Jane Wilhelms, "Collision detection and response for computer animation," in *Computer Graphics (SIGGRAPH '88 Proceedings)*, John Dill, Ed., Aug. 1988, vol. 22, pp. 289–298.

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

[8] 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. vol RA-4, pp. 193–203, 1988.

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

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

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

[12] S. Fisher and M. C. Lin, "Deformed distance fields for simulation of non-penetrating flexible bodies," *Proc. of EG Workshop on Computer Animation and Simulation*, 2001.

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

[14] M. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.

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

[16] M.C. Lin and John F. Canny, "Efficient algorithms for incremental distance computation," in *IEEE Conference on Robotics and Automation*, 1991, pp. 1008–1014.

[17] Brian Mirtich, "V-Clip: Fast and robust polyhedral collision detection," *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 177–208, July 1998.

[18] S. Ehmann and M. C. Lin, "Accelerated proximity queries between convex polyhedra using multi-level Voronoi marching," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

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

[20] 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*, pp. 322–331, 1990.

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

[22] Philip M. Hubbard, "Collision detection for interactive graphics applications," *IEEE Trans. Visualization and Computer Graphics*, vol. 1, no. 3, pp. 218–230, Sept. 1995.

[23] 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.*, vol. 4, no. 1, pp. 21–36, 1998.

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

[25] S. Ehmann and M. C. Lin, "Accurate and fast proximity queries between polyhedra using convex surface decomposition," *Computer Graphics Forum (Proc. of Eurographics'2001)*, vol. 20, no. 3, 2001.

[26] Leonidas J. Guibas and R. Seidel, "Computing convolutions by reciprocal search," in *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, 1986, pp. 90–99.

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

[28] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha, "Fast 3D geometric proximity queries between rigid and deformable models using graphics hardware acceleration," Tech. Rep., UNC-CS, 2002.

[29] E.G. Gilbert and C.J. Ong, "New distances for the separation and penetration of objects," in *Proceedings of International Conference on Robotics and Automation*, 1994, pp. 579–586.

[30] M. E. Houle and G. T. Toussaint, "Computing the width of a set," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-10, no. 5, pp. 761–765, 1988.

[31] J. Schwerdt, M. Smid, J. Majhi, and R. Janardan, "Computing the width of a three-dimensional point set: an experimental study," Report 18, Department of Computer Science, University of Magdeburg, Magdeburg, Germany, 1998.

[32] Bernard Chazelle, H. Edelsbrunner, Leonidas J. Guibas, and Micha Sharir, "Diameter, width, closest line pair, and parametric searching," in *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 1992, pp. 120–129.

[33] Pankaj K. Agarwal and Micha Sharir, "Efficient randomized algorithms for some geometric optimization problems," in *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 326–335.

[34] Leonidas J. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams," *ACM Trans. Graph.*, vol. 4, no. 2, pp. 74–123, Apr. 1985.

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

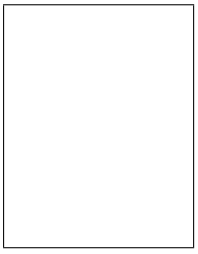
[36] J. O'Rourke, C.-B. Chien, T. Olson, and D. Naddor, "A new linear algorithm for intersecting convex polygons," *Comput. Graph. Image Process.*, vol. 19, pp. 384–391, 1982.

[37] B. Grünbaum, *Convex Polytopes*, John Wiley & Sons, New York, NY, 1967.

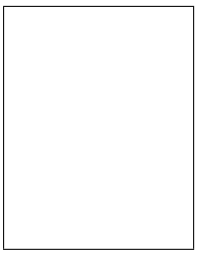
[38] Y. J. Kim, M. Otaduy, M. C. Lin, and D. Manocha, "Six-degree-of-freedom haptic display using localized contact computations," in *Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, March 2002.

[39] Bernard Chazelle, D. Dobkin, N. Shouraboura, and A. Tal, "Strategies for polyhedral surface decomposition: An experimental study," *Comput. Geom. Theory Appl.*, vol. 7, pp. 327–342, 1997.

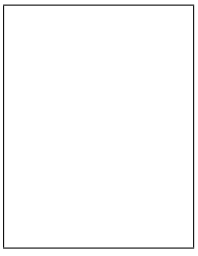
[40] Y. Kim, M. Otaduy, M. Lin, and D. Manocha, "Fast penetration depth computation for physically-based animation," in *ACM Symposium on Computer Animation*, 2002.



Young J. Kim is a postdoctoral researcher in the Department of Computer Science at the University of North Carolina (UNC) at Chapel Hill. He received his B.S. and M.S. degrees in Computer Science and Statistics in 1993 and 1996, respectively, from Seoul National University, and his Ph.D. degree in Computer Science in August 2000 from Purdue University. Since joining UNC-Chapel Hill, he has been developing geometric algorithms for interactive computer graphics including haptics and collision detection. His research interests include computer graphics, computational geometry, CAGD, interactive video games, and scientific visualization.



Ming C. Lin is an associate professor in the Department of Computer Science at the University of North Carolina at Chapel Hill. She received her B.S., M.S., and Ph.D. degrees in Electrical Engineering and Computer Science in 1988, 1991, and 1993, respectively, from the University of California at Berkeley. She has published more than 80 papers in physically-based modeling, robotics, computational geometry, computer graphics and virtual environments. She has served in the program committees of leading conferences in these areas. She received the NSF Young Faculty Career Award in 1995, Honda Research Initiation Award in 1997, UNC/IBM Junior Faculty Development Award in 1999, and UNC Hettleman Award for Scholarly Achievements in 2002. She has also received best papers awards at the Eurographics Conferences.



Dinesh Manocha is currently a professor of computer science at the University of North Carolina at Chapel Hill. He received his B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Delhi in 1987; and his M.S. and Ph.D. degrees in computer science from the University of California at Berkeley in 1990 and 1992, respectively. He has published more than 130 papers in the leading conferences and journals in computer graphics, geometric modeling, computational geometry, robotics and symbolic computation. He was selected an Alfred P. Sloan Research Fellow, received NSF Career Award in 1995 and Office of Naval Research Young Investigator Award in 1996, Honda Research Initiation Award in 1997, and Hettleman Prize for scholarly achievement at UNC Chapel Hill in 1998. He has also received best paper awards at the ACM SuperComputing, ACM Multimedia and Eurographics Conferences.