Fast Penetration Depth Estimation using Rasterization Hardware and Hierarchical Refinement

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

University of North Carolina, Chapel Hill, NC, USA
{youngkim,lin,dm}@cs.unc.edu
http://gamma.cs.unc.edu/~PD

Abstract. We present a novel and fast algorithm to estimate penetration depth (PD) between two polyhedral models. Given two overlapping polyhedra, it computes the minimal translational distance to separate them using a combination of discretized computations and hierarchical refinement. The algorithm computes pairwise Minkowski sums of decomposed convex pieces, performs closest point query using rasterization hardware, and refines the estimated PD by incremental walking. It uses bounding volume hierarchies, model simplification, and culling algorithms to further accelerate the computation and refines the estimated PD in a hierarchical manner. We highlight its performance on complex models.

1 Introduction

The problem of computing a distance measure between geometric objects arises in robotics, dynamic simulation, computer gaming, virtual environments, etc. It includes computation of minimal Euclidean or separation distance between disjoint geometric objects as well as a measure of penetration or intersection between two overlapping objects. The separation distance computation problem has been well-studied in the literature and a number of efficient and practical algorithms are known for polyhedral models. On the other hand, there is relatively less work on penetration depth computation between two intersecting objects

Given two inter-penetrating rigid polyhedral models, the penetration measure between them can be defined using different formulations. One of the widely used measures for quantifying the amount of intersection is *penetration depth*, commonly defined as the minimum translational distance required to separate two intersecting rigid models [5,6,8]. Penetration depth (PD) is often used in motion planning [15], contact resolution for dynamic simulation [24,26,30], force computation in haptic rendering [12,25], tolerance verification for virtual prototyping [29], etc.

The PD between two overlapping objects can be formulated based on their *Minkowski sum*. Given two polyhedral models, say P and Q, the PD corresponds to the minimum distance from the origin of the Minkowski sum, $P \oplus (-Q)$, to the surface of this sum. However, the computational complexity of computing the Minkowski sum can be $O(n^6)$, where n is the number of features [8]. In addition to its high computational complexity, the resulting algorithms are also susceptible to accuracy and robustness problems. Hence, no practical algorithms are currently known for accurately computing the PD between general polyhedral models.

Main Results: We present a novel approach to estimate the PD between general polyhedral models using a combination of discretized computations and hierarchical representations. Given the global nature of the PD problem, we systematically decompose the boundary of each polyhedron into convex pieces, compute the pairwise Minkowski sums of the resulting convex polytopes, and use graphics rasterization hardware to perform the closest point query up to a given discretized resolution. The results obtained are refined using a local walking algorithm. To further speed up this computation and improve the estimate, we present a hierarchical refinement technique that takes advantage of geometry culling, model simplification, and local refinement with greedy walking. The overall approach combines discretized closest point queries with geometry culling and refinement at each level of the hierarchy. We also analyze its performance as a function of discretization error.

The resulting algorithm has been implemented and tested on different benchmarks. Depending on the combinatorial complexity of polyhedra and their relative configuration, its performance varies from a fraction of a second to a few seconds on a 1.6 GHz PC with an nVidia GeForce 3 graphics card.

2 Previous Work

In this section, we briefly review previous work related to proximity queries, penetration depth computation, and the use of discretized computations on a geometry SIMD machine for geometric applications.

2.1 Collision and Distance Queries

The problems of collision detection and distance computations are well studied in computational geometry, robotics, and simulated environments. Check out [22] for a survey.

2.2 Penetration Depth Computation

A few efficient algorithms to compute the penetration depth (PD) between convex polytopes have been proposed. The simplest exact algorithm is based on computing their Minkowski sum [13,16] followed by computing the closest point to its boundary from the origin. But its worst case complexity is $O(mn)$, where m and n are the number of features in each polytope. Dobkin et al. computed the directional PD using Dobkin and Kirkpatrick polyhedral hierarchy [8]. For any direction d, it finds the directional PD in $O(\log n \log m)$ time. A randomized algorithm to compute the PD is given in [1]. 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.

Given the worst-case $O(mn)$ complexity of PD computation between convex polytopes, a number of approximation approaches have been proposed for interactive applications. All of them either compute a subset of the boundary or a simpler approximation of the Minkowski sum and compute an upper or lower bound to the PD [4,5,18,27]. Some of them also take advantage of frame-to-frame coherence and perform incremental computations.

Other approximation approaches for general polygonal models are based on discretized distance fields. These include algorithms based on fast marching level-sets for 3D models [10] and others based on graphics rasterization hardware and multi-pass rendering for 2D objects [14].

2.3 Hardware-Assisted Algorithms for Geometric Applications

Interpolation-based polygon rasterization hardware is increasingly being used for geometric applications. A recent survey on different applications is given in [31]. The rasterization hardware performs discretized computations and the resolution is typically determined by the number of pixels in the frame-buffer or depth-buffer. The main benefit comes from speed and dealing with relatively few degenerate configurations.

3 Background and Overview

In this section, we give a brief overview of the PD computation problem and our approach to solve it.

3.1 Penetration Depth and Minkowski Sums

Let P and Q be two intersecting polyhedra. The PD of P and Q, $PD(P,Q)$, is the minimum translational distance that one of the polyhedra must undergo to render them disjoint. Formally, $PD(P,Q)$ is defined as:

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

The Minkowski sum, $P \oplus Q$, is defined as a set of pairwise sums of vectors from P and Q. In other words, $P \oplus Q = \{\boldsymbol{p}+\boldsymbol{q} |\ \boldsymbol{p} \in P, \boldsymbol{q} \in Q\}$. Similarly, $P \oplus (-Q)$ can be defined as $P \oplus (-Q) = \{\boldsymbol{p}-\boldsymbol{q} |\ \boldsymbol{p} \in P, \boldsymbol{q} \in Q\}$.

A general framework to compute the PD is based on Minkowski sums. Without loss of generality, let us assume that two polyhedra P and Q are defined with respect to the global origin $\boldsymbol{O}$. If two polyhedra P and Q intersect, then the origin $\boldsymbol{O}$ is inside $P \oplus -Q$. The $PD(P,Q)$ is defined as a minimum distance from $\boldsymbol{O}$ to the surface of $P \oplus -Q$ [5].

It is relatively easier to compute Minkowski sums of convex polytopes as compared to general polyhedral models. One possible approach for computing Minkowski sums for general polyhedra is based on *decomposition*. It uses the following property of Minkowski computation. If $P = P_1 \cup P_2$, then $P \oplus Q = (P_1 \oplus Q) \ \cup \ (P_2 \oplus Q)$. The resulting algorithm combines this property with convex decomposition for general polyhedral models:

1. Compute a convex decomposition for each polyhedron

2. Compute the pairwise convex Minkowski sums between all possible pairs of convex pieces in each polyhedron
3. Compute the union of pairwise Minkowski sums.

After the second step, there can be $O(n^2)$ pairwise Minkowski sums and their union can have $O(n^6)$ complexity [2].

This approach provides an algorithmic framework to compute the Minkowski sum. However, its practical utility is unclear. Besides the combinatorial complexity, it is a major challenge to have a robust implementation of an algorithm for union computations in 3D.

3.2 Our Approach

Our algorithm to estimate the PD is based on the decomposition approach described in Section 3.1. In order to overcome its combinatorial and computational complexity, we use a *surface-based* convex decomposition of the boundary and perform discretized computations and local walking to estimate the PD. We do not explicitly compute the boundary of the union or any approximation to it. Rather, we perform the *closest point query* using hardware-assisted massive ray shooting that estimates the closest point from the origin to the boundary of the union of pairwise Minkowski sums. The resulting maximum depth fragment at each pixel computes an approximation to the PD, up to the pixel resolution used for this computation. Given this PD estimate, we further refine it using an incremental algorithm that performs a local walk on the Minkowski sum. Each step of our approach is relatively simple to implement. However, its worst case complexity can be as high as $O(n^4)$ because of the number of pairwise Minkowski sums and the computational complexity of the closest point query.

We improve the performance of the algorithm using a number of acceleration techniques. These include hierarchical representation based on convex bounding volumes, use of model simplification algorithms, and geometry culling approaches applied to both Minkowski sum computation and hardware assisted ray-shooting. These are explained in detail in Section 5.

3.3 Notation

We use bold-faced letters to distinguish a vector from a scalar value (e.g. the origin, $\boldsymbol{O}$). In Table 1, we enumerate the notations that we use throughout the paper.

4 Penetration Depth Computation

In this section, we present our algorithm for estimating global PD.

4.1 Object Decomposition

We decompose the boundary of each polyhedron P into a collection of convex patches c_i. These c_i's are mutually disjoint, and the union of all the c_i's covers

Notation	Meaning
∂P	The boundary of P
C_i^P	A decomposed convex piece of P
$C_i^{P,l}$	A decomposed convex piece of P at level l
M_{ij}	Minkowski sum between C_i and C_j
d_{est}^k	kth refinement of the PD estimation

Table 1. Notation Table

the entire boundary of P, ∂P. Computing an optimal decomposition is NP-hard and different heuristics to compute such decompositions are well-known [7,9].

Furthermore, we compute a convex hull of each surface patch, c_i, and denote the resulting polytope by C_i. The union of these C_i's completely covers the boundary of the original polyhedron P. Notice that our decomposition strategy is merely a partition of ∂P, not of P. This surface decomposition is sufficient for PD computation, because we are only concerned with the surface of Minkowski sums between polyhedra.

4.2 Pairwise Minkowski Sum Computation

Our PD computation algorithm is based on the decomposition approach described in Section 3.1. The first step involves computing the pairwise Minkowski sums between all possible pairs of convex polytopes, C_i^P and C_j^Q, belonging to P and Q, respectively. Let us denote the resulting Minkowski sum as M_{ij}. Various algorithms are known for computing Minkowski sums of convex polytopes. Using the *topological sweep*, we can compute M_{ij} in $O(n \log n + k)$ time, where n is the number of features in C_i^P and C_j^Q and k is the number of features in M_{ij} [13].

4.3 Closest Point Query Using Rasterization Hardware

Given all the pairwise Minkowski sums, M_{ij}, let

$$M = \bigcup_{ij} M_{ij}. \tag{2}$$

Our goal is to compute the closest point on the boundary of M, i.e. ∂M, from the origin. We perform discretized computations to estimate the closest point on ∂M. It involves using polygon rasterization hardware to perform this query up to a given pixel resolution. The main idea is to rasterize ∂M from the origin without computing a surface representation of ∂M explicitly. After that we compute the closest point, distance and direction.

Rasterization Hardware The discretized computations are performed on two dimensional pixel data in parallel. The underlying model corresponds to an abstract *geometric SIMD machine* which switches between different states [21]. Typical instruction sets allowed in the geometric SIMD model include simple arithmetic, logical and relational operations on the pixel data, also known as the *frame buffer*. Current graphics rasterization hardware is a close realization of this model. From the viewpoint of our PD algorithm, the rasterization hardware is a multi-processor system that performs massive ray-shooting operations onto geometric primitives in parallel. We take advantage of these massive ray-shooting functionality of the rasterization hardware to perform the closest point query.

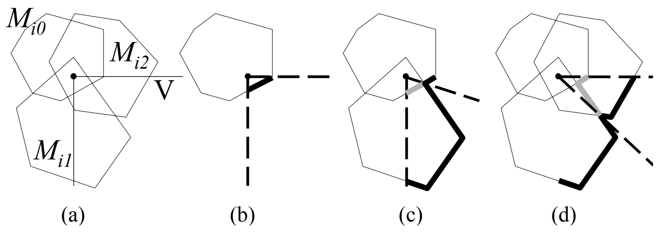

Fig. 1. Rendering the Boundary of the Union From Inside. *In (a), V is the current view-frustum. In (b), M_{i0} is rendered, and a new ∂M is constructed (thick line). In (c), when M_{i1} is rendered, it opens up a new window (dotted line), and the update region (thick gray line) on the current ∂M is established. Thus a new ∂M (thick line) is constructed. In (d), we perform the same procedure for M_{i2}.*

Rasterizing the Boundary of the Union Our algorithm for rasterizing ∂M from a point inside is essentially a massive ray-shooting procedure from the origin to ∂M by incrementally expanding the front of ∂M. The algorithm can require m^2 passes, where m is the number of convex polytopes, M_{ij}.

The algorithm maintains the current boundary of M, ∂M^k, where k is the current iteration, and incrementally expands it with M_{ij} that intersects ∂M^k. We attempt to add M_{ij} by drawing the front faces of M_{ij}. The front faces that "pierce" the current ∂M^k open up a window through which the origin can see ∂M. After that we draw the backfaces of M_{ij} into the opened window using the maximum depth test.

Computing the Closest Point For a given view, we can compute the closest point on the boundary by simply finding the pixel with the minimum distance value. The algorithm transforms the pixel depth values into distance values based on their (x, y) coordinate positions on the viewing plane. Each pixel depth value is divided by $\cos\theta$, where θ is the angle between the vector to the (x, y) position on the viewing plane and the center viewing direction. The minimum distance and direction to the closest point are derived from the pixel position containing the minimum transformed depth value. In order to examine views in all directions, we construct six views on the faces of a cube around the origin and repeat the operation. For more information about the closest point query, we refer the readers to see our companion paper [17].

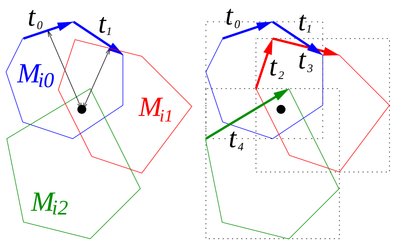

Fig. 2. Local Refinement by Walking. *We refine the PD by iteratively minimizing the distance between the origin and a line (a triangle in 3D) on the Minkowski sum M_{i0}. Thus, in (a), there is a transition from t_0 to t_1, since the distance from the origin to t_1 is smaller than that to t_0. In (b), the feature t_3 can reduce the PD even further, the transition of the Minkowski sum from M_{i0} to M_{i1} is followed.*

4.4 Local Refinement

The accuracy of the closest point query and PD estimate is limited by the pixel resolution of the rasterization hardware. We further refine it and improve the PD estimate by performing local *walks* on the boundary of the Minkowski sum of P and Q.

Initially the refinement algorithm starts with identifying the features of P and Q that contribute to the current PD estimate. Each triangle in M_{ij} is generated by only three possible sets of feature combinations from P and Q. These include vertex/face (VF), face/vertex (FV) and edge/edge (EE) combinations [13], and we use that relationship to compute the actual PD features from each polyhedron that correspond to the current PD estimate. At any time, the algorithm also maintains a notion of *current-Minkowski-sum*, which contains the current PD features.

Once the PD features and the Minkowski sum (M_{ij}) which contains them have been identified, the algorithm refines the current PD estimate by locally walking on the surface of M_{ij}, the current-Minkowski-sum. This walk proceeds by iteratively minimizing the distance from the origin to the surface of M_{ij}. We repeat this process until the algorithm reaches a local or global minimum.

As shown in Fig. 2 the algorithm needs to avoid features that are inside the volume of other Minkowski sums. Although it walks towards the interior of the volume, it sets the current-Minkowski-sum accordingly. Therefore, each time the algorithm is walking, it keeps track of which M_{ij}'s might intersect with the current PD features. We accomplish this by keeping track of a subset of Minkowski sums that can potentially intersect with the current PD features and the current-Minkowski-sum.

Let us denote the current-Minkowski-sum as M_{ij}, and also denote the subset of Minkowski sums that potentially intersect with M_{ij} as M_{ij_0}, M_{ij_1}, ...,

M_{ij_l}. Here, we conservatively determine M_{ij_l}'s by intersection checks based on an axis-aligned bounding box (AABB) of the Minkowski sum. Moreover, for each M_{ij_l}, we also keep track of a closest triangle t_{k_l} to the current PD feature t_k in M_{ij}. The overall refinement algorithm proceeds as:

1. Let the triangle t_k in M_{ij} correspond to the PD features computed based on the closest point query. Compute the set of Minkowski sums $M_{ij_0}, M_{ij_1}, ..., M_{ij_l}$ that intersect M_{ij} based on checking their AABBs for overlap. Also compute $t_{k_0}, t_{k_l}, ..., t_{k_l}$, which is a set of triangles respectively on M_{ij_l}'s that is closest to t_k on M_{ij}.
2. Identify the triangles incident to t_k on M_{ij}.
3. Find a neighboring triangle, say t_{k+1}, that results in maximum decrease in the PD estimate and does not intersect with t_{k_l}'s. Change the current PD features from t_k to t_{k+1}. Also update t_{k_l} on each M_{ij_l} to the closest feature to t_{k+1}.
4. If step 4 fails, check whether there exists t_{k_l} in M_{ij_l} such that it intersects with the triangles incident to t_k or t_k itself but reduces the PD. If it exists, repeat the walk from step 1 by setting t_{k_l} as t_k and M_{ij_l} as M_{ij}.
5. Repeat the steps 2-4 until there is no more improvement in the PD.

Eventually the algorithm computes a local minimum on the boundary of the Minkowski sum, M.

5 Acceleration Techniques

The global PD computation algorithm described in Section 4 computes an upper bound on the amount of PD between two polyhedral models. However, its running time can vary based on the underlying models as well as their relative configuration. In this section, we present a number of acceleration techniques to improve its performance. These include hierarchical culling, model simplification, and geometry culling for closest point query.

5.1 Geometry Culling

A significant fraction of the time of the PD estimation algorithm is spent in pairwise Minkowski sum computation. The algorithm presented in Section 4.2 considers all pairs of convex polytopes, C_i^P and C_j^Q, and computes their Minkowski sum, M_{ij}. If we are given an upper bound on the PD, d_{est}, we can eliminate some pairs of convex polytopes without computing their Minkowski sum. This is based on the following lemma:

Lemma 1. *Let d_{ij} be the separation or Euclidean distance between C_i^P and C_j^Q. If $d_{ij} > d_{est}$, then the closest point from the origin to ∂M lies on $\partial(M - M_{ij})$.*

Based on the Lemma 1, we can cull away all pairs of convex polytopes, C_i^P and C_j^Q, whose separation distances are more than d_{est}. Computing separation distance between convex polytopes is relatively cheap as compared to

Minkowski sum computation and a number of efficient algorithms are known [5,23]. The efficiency of this culling approach depends on the quality of the estimate, d_{est}. Furthermore, checking all possible pairs for separation distance can take $O(n^2)$ time. We improve their performance using a bounding volume hierarchy to perform hierarchical culling.

5.2 Bounding Volume Hierarchy

We compute a bounding volume (BV) hierarchy for each polyhedron using a convex polytope as the underlying BV. Each convex polytope obtained using the decomposition algorithm explained in Section 4.1 becomes a leaf node in the hierarchy. We recursively compute the internal nodes in a bottom-up manner, by merging the children nodes and computing the convex hull of the union of their vertices. Let us define the nodes of polyhedron P at level l as $C_i^{P,l}$. The resulting hierarchy is a hierarchy of convex hulls.

This hierarchy is used in our runtime algorithm to speed up the intersection and separation distance queries for the culling algorithm. Furthermore, each level of the hierarchy provides an approximation of the model, which is used by the PD estimation algorithm.

5.3 Hierarchical Culling

We use the BV hierarchy to speed up the performance of the object-space culling algorithm. The goal is to start with an initial estimate to the PD and refine it at every level of the tree. We denote the estimate computed using level k of each BV tree as d_{est}^k.

We initially start with the root nodes of each hierarchy, $C_0^{P,0}$ and $C_0^{Q,0}$, which correspond to the convex hulls of P and Q, respectively. We compute the PD between those convex polytopes [5,4,18] and use that as the estimated PD at level 0. The algorithm proceeds in a hierarchical manner through the levels in each tree:

1. Consider all the pairwise nodes at level k in each tree, $C_i^{P,k}$ and $C_j^{Q,k}$. For each (i, j) pair, compute the separation distance between them. If the nodes overlap, the separation distance is zero.
2. Discard all the node pairs whose separation distances are more than d_{est}^k. Compute the Minkowski sum for the rest of the pairs.
3. Perform the closest point query on the Minkowski sum pairs and compute the new PD estimate, d_{est}^{k+1} using rasterization hardware.
4. Refine the estimate, d_{est}^{k+1} using the object space walking algorithm presented in Section 4.4.

During each iteration, we go down a level in each tree. If we reach the maximum level in one of the trees, we do not traverse down in that tree any further. The algorithm computes an upper bound on the PD in an iterative manner and refines the bound with every traversal as: $d_{est}^0 \geq d_{est}^1 \geq \ldots \geq d_{est}^h$, where h is the maximum height. Finally, the algorithm returns d_{est}^l as the estimated PD between P and Q.

5.4 Model Simplification

Some internal nodes of the hierarchy may have a high number of vertices and that affects the complexity of pairwise Minkowski sum computation. We pre-compute a single convex simplification for each internal node in the BV tree. The simplifications at each level of the BV tree provide a low polygon count approximation to the original models. We compute a simplification for each internal node in the following manner:

1. Simplify the node using any simplification error metric.
2. Compute the convex hull of each simplified node.
3. Scale the resulting convex polytope to enclose the internal node or the underlying geometry as tightly as possible.

We use the simplified BVs to improve the performance of the computations in step 2 (pairwise Minkowski sum computation) and step 3 (closest point query) of the hierarchical culling algorithm presented in Section 5.3. The simplified BVs can increase the estimated PD value, d_{est}, as compared to the original nodes computed by the BV hierarchy computation algorithm. As a result, the number of pairwise Minkowski sums that can be culled at intermediate levels of the hierarchy based on d_{est} may be reduced. However, the running time of the algorithm is significantly reduced. Also, it does not change the accuracy of the final result, as the algorithm does not simplify the leaf nodes in the BV tree.

5.5 Culling for Closest Point Query

The algorithm also spends a considerable fraction of its time in performing the closest point query using the rasterization hardware (as described in Section 4.3). Here we present a number of techniques to improve its performance.

First of all, we compute a subset of the pairs, M_{ij}'s, that contain the origin and render them only once in the algorithm described in Section 4.3. All the pairwise Minkowski sums in this subset have a zero *hop*. We identify this subset, say l out of total of m pairs of M_{ij}'s, by checking whether the corresponding convex polytopes, C_i^P and C_j^Q, overlap [5,9,23]. Once we have computed these l M_{ij}'s, we first render them using the maximum depth test and then the remaining $(m-l)$ pairwise Minkowski sums, M_{ij}'s, $(m-l)$ times using the incremental algorithm.

Secondly, when we repeat the closest point query six times, once for each face of the cube, we apply a culling technique similar to the one discussed in Section 5.1. At each view, the algorithm maintains the current minimum depth value, d_{est}, and then as it proceeds to the next view, it culls away the M_{ij}'s whose distance from the origin is more than d_{est}, as shown in Lemma 1. Finally, for each query, when we render the M_{ij}'s, we perform view-frustum culling by checking whether the axis aligned bounding box of each M_{ij} lies in the current view. This view frustum culling significantly reduces the number of primitives rendered during each iteration of the algorithm.

6 Analysis of PD algorithm

In this section, we analyze the performance of our PD algorithm and discuss its accuracy.

6.1 Performance Analysis

The basic PD algorithm presented in Section 4 has the following computational complexity at run-time:

- Each non-convex object can have $O(n)$ convex pieces using the convex surface decomposition presented in Section 4.1. Thus, each convex piece has $O(C)$ complexity on the average. In practice, C is a small number of less than 10.
- The pairwise Minkowski sum computation has an input of n^2 combinations of convex pieces from two non-convex objects, and each M_{ij} computation requires $O(C \log C)$ time. Therefore, the overall pairwise Minkowski sum computation requires $O(n^2 C \log C)$ running time.
- The closest point query requires m^2 iterations, where $m = n^2$ is the number of M_{ij}'s. Each iteration requires rasterizing $O(C^2)$ triangles in the worst case, and we assume that the triangle rasterization takes constant time T. The transformation for the perspective correction requires $O(RW^2)$ time, where W is the pixel resolution and R is a cost of a single *read-back* from the frame buffer. Therefore, the total computational complexity of the closest point query is $O(TC^2 n^4 + RW^2)$.
- Each refinement walk step requires $O(C^2 n^2)$ time in the worst case, since it needs to keep track of all the potential intersectors of the current-Minkowski-sum. In practice, each step requires a small number of constant iterations as opposed to the worst case complexity.
- In summary, the object space computation requires $O(n^2 C \log C)$ time, and the image space computation (i.e. closet point query) requires $O(TC^2 n^4 + RW^2)$ time.

The performance of the basic algorithm is improved by different techniques highlighted in Section 5. However, the performance of the resulting algorithm using hierarchical refinement depends heavily on the extent of object-space culling, which is directly related to the amount of inter-penetration between the objects. As a result, when the penetration between two polyhedra is relatively shallow, the algorithm is able to cull away a very high percentage of Minkowski pairs (as shown in Table 2 in Section 7.2) and quite fast in practice. However, it is very hard to analyze the culling performance quantitatively, since the performance depends on various parameters of objects such as its complexity, aspect ratios, the amount of interpenetration between the objects, and their relative configuration.

6.2 Error Analysis

Our algorithm always computes an upper estimate to the PD. In other words, the algorithm may be conservative and the computed answer may be more than the global minimum defined in Equation 1. The rasterization errors and precision of discretized computations governs the tightness of the resulting answer. The main sources of these errors are as follows:

1. The discretization of ray directions to lie on a pixel grid for each view.
2. The fixed precision of the Z-buffer.

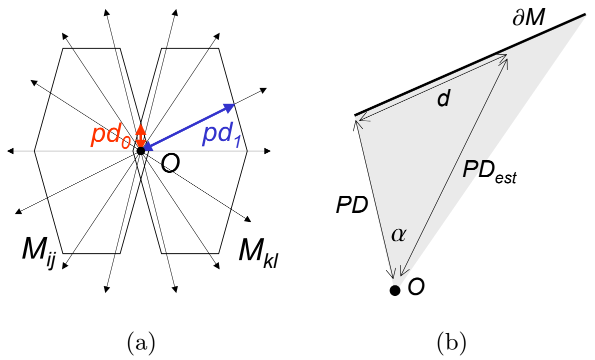

Fig. 3. (a) Ray Shooting Error. *The true PD value is* pd_0*, which is the minimum distance from* O *to* $\partial(M_{ij} \cup M_{kl})$*. However, due to the discretized ray shooting, the reported PD value* pd_1 *can be arbitrarily larger than* pd_0. (b) Error Bound. *PD is optimal, and PD_{est} is closest to PD computed by our algorithm. d is the upper bound of the length of an edge in ∂M. Then,* $PD_{est} \leq PDcos\alpha + \sqrt{d^2 + PD^2 sin^2\alpha}$.

Increasing the resolution of the grid decreases the possibility of the worst-case angular error. Moreover, constructing tighter bounds on the minimum and maximum distances in each view (near and far plane distances) decreases the Z-buffer precision error. However, as illustrated in Fig. 3-(a), the worst case error can be arbitrarily large regardless of the resolution of the grid.

In practice, we can assume that the massive ray-shooting assisted by rasterization hardware is dense enough that every face in M is hit by at least one ray. Furthermore, since we explicitly compute M_{ij}, we know the upper bound d of the length of an edge in ∂M. In this case, as shown in Fig. 3-(b), the upper bound on the estimate, PD_{est} is:

$$PD_{est} \leq PD\ cos\alpha + \sqrt{d^2 + (PD)^2\ sin^2\alpha}$$

by using the cosine law, where α is the smallest angle between rays and PD is the optimal penetration depth between the underlying polyhedra.

We also observed that the PD value is rapidly converging as the pixel resolution increases. The convergence rate also depends on the relative configuration between the objects. Fig. 4 shows a typical convergence behavior of our PD algorithm. The figure shows the convergence rates of two intersecting tori in different configurations, touching and interlocked. More specific data about these models and their relative configuration are given in Fig. 5

and Table 2 in Section 7.2. In practice, we are able achieve a reasonably converging PD value when the pixel resolution is set to 256 × 256.

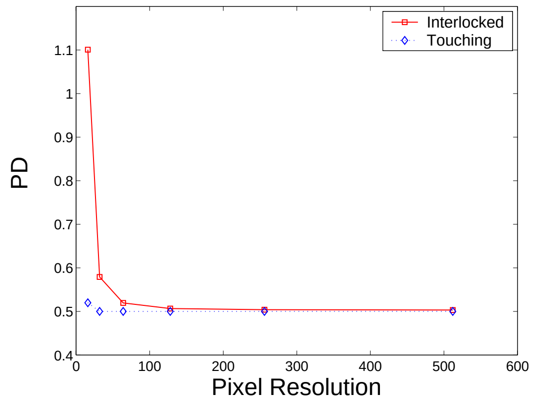

Fig. 4. PD Convergence Rate With Respect To the Pixel Resolution. Two different configurations of two intersected tori, touching and interlocked, are given.

7 Implementation and Results

In this section, we describe the implementation of our PD computation algorithm and demonstrate its performance on different benchmarks and applications. We also refer the readers to see [19,20] where we have used our PD computation algorithm for dynamic simulation of rigid bodies and tolerance verification for rapid prototyping of complex structures.

7.1 Implementation Issues

We use the SWIFT++ implementation of the *Voronoi marching* technique [9] to efficiently perform the separation distance query. It performs distance queries between non-convex polyhedra by using a hierarchy of convex hulls. We use the public domain QHULL package [3] for convex hull computation in 3D. QHULL is particularly efficient for dealing with a relatively small number of points, which is the case in our algorithm. We use the QSlim implementation [11] of the quadric error metric simplification algorithm to ensure that the intermediate nodes of the bounding volume trees do not have more than 50 vertices.

We implement the closest point query operation using OpenGL graphics library. Also, we typically set the screen space resolution to 128×128 at the intermediate step of the hierarchical refinement, then at the finest level of the refinement, we set the resolution to 256×256. For our benchmarking models, these different resolution schemes provide us with results of a reasonable accuracy, and they also balance the computation time between the object space and the image space.

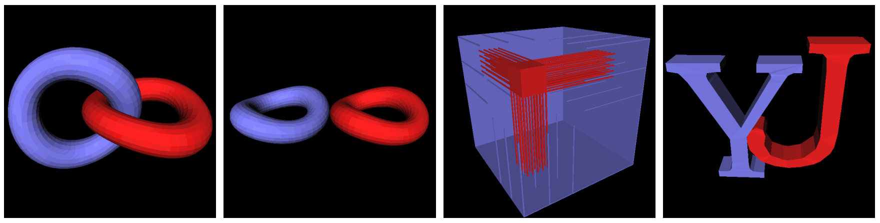

Fig. 5. PD Benchmark Models. *From left to right: interlocked tori, touching tori, interlocked grates, and letters.*

7.2 Benchmark Results

We benchmark our PD algorithm with four models: interlocked tori, touching tori, interlocked “grates” and a pair of alphabet models, with their relative configuration shown in Fig. 5. We used the tori models because it is relatively difficult to compute a good convex decomposition for them. The interlocked “grates” model was chosen because the combinatorial complexity of its exact Minkowski sum is $O(m^3n^3)$ [28]. In our benchmarks, m and n are 1134 and 444, respectively. Therefore, it is a very challenging scenario for any PD computation algorithm. Earlier approaches based on localized computations or convex volumetric decomposition are unable to compute the PD efficiently and accurately on these benchmarks.

We measure the timings on a PC equipped with an Intel Pentium IV 1.6 GHz processor, 512 MB main memory and GeForce 3 graphics card. The complexity of the models varies from a few hundred faces to a few thousand faces. The number of leaf nodes, computed using the convex surface decomposition algorithm, vary from 67 pieces to 409 pieces. The running times vary based on the model complexity and the relative configuration of two polyhedra. It can vary from a fraction of a second, for the touching tori and a pair of alphabet models, to a few seconds for models that have deep penetration (e.g. interlocked tori and interlocked “grates”). Most of the time is spent in pairwise Minkowski sum computations and closest point queries using the graphics hardware. The local refinement based on the walking algorithm is quite fast and takes only a few milliseconds. Detailed timings for some levels of the hierarchy are given in Table 2. The acceleration techniques and hierarchical refinement result in several orders of magnitude improvement in the overall running time. Furthermore, the algorithm is able to compute accurate PD estimates in these cases.

7.3 Performance Speedup by Acceleration Techniques

In Table 3, we have compared the performance of our accelerated PD algorithm presented in Section 5 with the basic algorithm presented in Section 4. As the table illustrates, the basic algorithm suffers from $O(n^4)$ computational costs, and our accelerated algorithm outperforms it by several orders of magnitude. The result is even more dramatic in a very complex scenario such as the interlocking grates model.

Level	Cull Ratio	Min. Sum	HW Query	d_{est}
3	31.2 %	0.219 sec	0.220 sec	0.99
5	96.7 %	0.165 sec	0.146 sec	0.53
7	98.3 %	1.014 sec	1.992 sec	0.50

(a) Interlocked Tori (2000 faces, 67 convex pieces each)

Level	Cull Ratio	Min. Sum	HW Query	d_{est}
3	98.4 %	0.135 sec	0.014 sec	0.29
7	99.9 %	0.105 sec	0.032 sec	0.29

(b) Touching Tori (2000 faces, 67 convex pieces each)

Level	Cull Ratio	Min. Sum	HW Query	d_{est}
3	0 %	0.66 sec	0.29 sec	6.41
7	96.9 %	0.43 sec	0.39 sec	0.63
9	99.9 %	0.03 sec	0.07 sec	0.63

(c) Grates (444 & 1134 faces, 169 & 409 pcs)

Level	Cull Ratio	Min. Sum	HW Query	d_{est}
2	50.0 %	0.055 sec	0.021 sec	0.06
4	56.2 %	0.099 sec	0.062 sec	0.03
6	97.6 %	0.080 sec	0.161 sec	0.01

(d) Alphabets (144 & 152 faces, 42 & 43 pcs)

Table 2. Benchmark Results. *We show the performance of our PD algorithm for various models. We also break down the performance to the object space culling rate, the pairwise Minkowski computation time and the closest point query time on some of the levels of the hierarchy.*

Type	Without Accel.	With Accel.
Interlocked Tori	4 hr	3.7 sec
Touching Tori	4 hr	0.3 sec
Grates	177 hr	1.9 sec
Alphabets	7 min	0.4 sec

Table 3. Performance Speedup by Acceleration Techniques

8 Summary and Future Work

We present a fast, global algorithm to estimate penetration depth between polyhedra using both image-space acceleration techniques and object-space culling and refinement algorithms. The resulting algorithm has been tested on difficult benchmarks.

There are several areas for future work. The performance of our algorithm can be further improved by exploring more optimizations. These include faster implementations of the closest point query using new features of the high-end graphics cards, as well as better hierarchical decompositions. Currently our algorithm only computes the minimum translational distance to separate two overlapping objects. It would be useful to extend it to handle rotational penetration depth.

9 Acknowledgments

This research was supported in part by ARO Contract DAAG55-98-1-0322, DOE ASCII Grant, NSF NSG-9876914, NSF DMI-9900157, NSF IIS-982167, NSF ACR-0118743, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, and Intel. We will like to thank Kenneth Hoff, Miguel A. Otaduy, and Dan Halperin for useful discussions and applications of this algorithm to dynamic simulation and virtual prototyping.

References

1. 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.
2. Boris Aronov, Micha Sharir, and Boaz Tagansky. The union of convex polyhedra in three dimensions. *SIAM J. Comput.*, 26:1670–1688, 1997.
3. B. Barber, D. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hull. Technical Report GCG53, The Geometry Center, MN, 1993.
4. G. Bergen. Proximity queries and penetration depth computation on 3D game objects. *Game Developers Conference*, 2001.
5. S. Cameron. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.
6. S. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 591–596, 1986.
7. Bernard Chazelle, D. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. *Comput. Geom. Theory Appl.*, 7:327–342, 1997.
8. D. Dobkin, J. Hershberger, D. Kirkpatrick, and Subhash Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.
9. S. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001)*, 20(3), 2001.
10. 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.
11. M. Garland and P. Heckbert. Surface simplification using quadric error bounds. *Proc. of ACM SIGGRAPH*, pages 209–216, 1997.
12. 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.
13. L. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom*, 2:175–193, 1987.

14. 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.
15. 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.
16. A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPS. *Computer and Graphics*, 16:107–116, 1992.
17. Y. Kim, K. Hoff, M. Lin, and D. Manocha. Closest point query among the union of convex polytopes using rasterization hardware. *Journal of Graphics Tools*, 2003. to appear.
18. Y. Kim, M. Lin, and D. Manocha. DEEP: Dual-space Expansion for Estimating Penetration depth between convex polytopes. In *IEEE Conference on Robotics and Automation*, 2002.
19. Y. Kim, M. Otaduy, M. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. In *ACM Symposium on Computer Animation*, 2002.
20. Y. Kim, M. Otaduy, M. Lin, and D. Manocha. Fast penetration depth computation using rasterization hardware and hierarchical refinement. Technical report, UNC-Chapel Hill TR02-014, 2002.
21. S. Krishnan, N. Mustafa, and S. Venkatasubramanian. Hardware-assisted computation of depth contours. In *ACM-SIAM Symposium on Discrete Algorithms*, 2002.
22. M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.
23. M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.
24. 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.
25. W. McNeely, K. Puterbaugh, and J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. *Proc. of ACM SIGGRAPH*, pages 401–408, 1999.
26. B. Mirtich. Timewarp rigid body simulation. *Proc. of ACM SIGGRAPH*, 2000.
27. C. J. Ong and E.G. Gilbert. Growth distances: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation*, 12(6), 1996.
28. G. D. Ramkumar. *Tracings and Their Convolution: Theory and Applications.* PhD thesis, Standford, March 1998.
29. A.A.G. Requicha. Mathematical definition of tolerance specifications. *ASME Manufacturing Review*, 6(4):269–274, 1993.
30. D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal of Numerical Methods in Engineering*, 39:2673–2691, 1996.
31. T. Theoharis, G. Papaiannou, and E. Karabassi. The magic of the Z-buffer: A survey. *Proc. of 9th International Conference on Computer Graphics, Visualization and Computer Vision, WSCG*, 2001.