Fast Penetration Depth Computation for Physically-based Animation

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

Department of Computer Science
University of North Carolina at Chapel Hill
{*youngkim,otaduy,lin,dm*}*@cs.unc.edu*
http://gamma.cs.unc.edu/PD

Abstract

We present a novel and fast algorithm to compute penetration depth (PD) between two polyhedral models for physically-based animation. Given two overlapping polyhedra, it computes the minimal translation distance to separate them using a combination of object-space and image-space techniques. The algorithm computes pairwise Minkowski sums of decomposed convex pieces and performs a closest point query using rasterization hardware. It uses bounding volume hierarchies, object-space and image-space culling algorithms to further accelerate the computation and refines the estimated PD in a hierarchical manner. We demonstrate its application to contact response computation and a time-stepping method for dynamic simulation.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Algorithms, Object Hierarchies, Physically Based Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible Line/Surface Algorithms

Keywords: Collision detection, Graphics hardware, Image-space computations, Dynamics simulation, Geometric modeling, Robotics

1 Introduction

The need to perform fast proximity queries, including collision detection, tolerance checking, distance calculation and penetration computation, arises in numerous areas. Some applications include physically-based animation, planning of autonomous characters, computer games, and virtual environments. While several of these queries, such as collision detection and distance computation, have been extensively studied in the field, there is relatively little work on penetration computation that provides a measure of intersection or penetration between two overlapping models. Given two interpenetrating 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 [Dobkin et al. 1993; Cameron and Culley 1986; Cameron 1997].

Most of existing collision detection algorithms and systems do not handle inter-penetrations and current distance computation algorithms do not provide a continuous distance measure when two (non-convex) objects collide. This can induce numerical problems, e.g. instability or invalid results, in dynamic simulation. Furthermore, several commonly used techniques like penalty-based simulation methods often need to perform PD queries for imposing the non-penetration constraint for rigid body simulation [McKenna and Zeltzer 1990; Mirtich 2000; Stewart and Trinkle 1996]. Various heuristics, such as reducing the frequency of PD computation [McNeely et al. 1999] or estimating PD based on the last closest feature pairs [Mirtich 2000; Gregory et al. 2000], are sometimes used. But, the results can be inconsistent and inaccurate. Fast and reliable PD computation is important for robust and efficient simulation of dynamical systems.

The PD between two overlapping objects can be formulated based on their *Minkowski sum*. However, the worst case computational complexity of computing the Minkowski sum can be $O(n^6)$, where n is the number of features [Dobkin et al. 1993]. In addition to its high computational complexity, the resulting algorithms are also susceptible to accuracy and robustness problems. Hence, no practical algorithms are yet known for accurately computing the PD between general polyhedral models.

Main Results: We present a novel approach to approximate the PD between general polyhedral models using a combination of object-space and image-space techniques. 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 the polygon interpolation based rasterization hardware to perform the closest point query up to image-space resolution. To further speed up this computation and improve the estimate, we use a hierarchical refinement technique that takes advantage of object-space culling and image-space acceleration. The overall approach combines image-space accelerated queries with object-space culling and refinement at each level of the hierarchy.

This 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 1.6GHz PC with an NVIDIA GeForce 3 graphics card. To illustrate the effectiveness of our algorithm, we demonstrate its application to contact response computation and a time-stepping method for rigid-body dynamic simulation. To the best of our knowledge, this is the first practical algorithm for computing a reliable PD between general polyhedral models for physically-based animation and it works well for different challenging scenarios.

Organization: The rest of the paper is organized in the following manner. We give a brief summary of the related work in Section 2 and present some background material along with an overview of our approach in Section 3. Section 4 describes the underlying algorithm that uses a combination of object space and image space com-

putations. We present a number of acceleration methods in Section 5 to improve the overall performance. Section 6 describes its implementation and performance on different configurations. Section 7 highlights its application to physically-based animation, including penalty-based rigid-body simulation and time-stepping methods.

2 Previous Work

In this section, we briefly review previous work related to proximity queries, penetration depth computation and the use of graphics rasterization hardware for geometric computations.

2.1 Collision and Distance Queries

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

For convex polytopes, various techniques have been developed based on Minkowski sum [Cameron 1997; Gilbert et al. 1988] and feature tracking using Voronoi regions [Lin and Canny 1991; Mirtich 1998]. Some of these algorithms also utilize the spatial and temporal coherence between successive frames and perform incremental computations [Baraff 1992; Cameron 1997; Lin and Canny 1991; Mirtich 1998].

For general polygonal models, bounding volume hierarchies (BVHs) have been widely used for collision detection and separation (or Euclidean) distance queries. They localize the problem based on the "divide-and-conquer" paradigm. BVHs often differ based on the underlying bounding volume or traversal schemes. These include the OBB trees [Gottschalk et al. 1996], sphere trees [Hubbard 1995], k-dops [Klosowski et al. 1998], and convex hull-based trees that use surface based convex decomposition [Ehmann and Lin 2001]. Due to the global nature of the PD problem, none of them can be directly used for PD computation between non-convex models.

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 [Guibas and Seidel 1987; Kaul and Rossignac 1992] 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 the Dobkin and Kirkpatrick polyhedral hierarchy [Dobkin et al. 1993]. 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 [Agarwal et al. 2000].

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 [Cameron 1997; Bergen 2001; Ong and Gilbert 1996; Kim et al. 2002a]. They 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 discretized algorithms based on fast marching level-sets for 3D deformable models [Fisher and Lin 2001] and others based on graphics rasterization hardware and multi-pass rendering for 2D objects [Hoff et al. 2001]. These techniques provide a localized estimation that may not be correct. No good global algorithms are known for PD computation between general polyhedral models.

2.3 Graphics Hardware for Geometric Applications

Interpolation-based polygon rasterization hardware is increasingly being used for geometric applications. These include visibility and shadow computations, CSG rendering, proximity queries, morphing, object reconstruction etc. A recent survey on different applications is given in [Theoharis et al. 2001]. All these algorithms perform computations in a discretized space (i.e. the image-space) and their accuracy is governed by the underlying pixel resolution. The set of proximity query algorithms include cross-sections and interferences [Rossignac et al. 1992] and distance computations, including separation and local penetration estimation [Hoff et al. 1999; Hoff et al. 2001]. An algorithm to compute a discretized approximation to the convolution of general polyhedral models using the rasterization hardware is presented in [Kaul and Rossignac 1992]. Algorithms for direct rendering of CSG models based on graphics rasterization hardware have been presented in [Epstein et al. 1989; Goldfeather et al. 1986; Wiegand 1996]. They render the CSG hierarchies using multiple passes of clipping (i.e. stencil tests) and depth tests.

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:

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

Here, $\boldsymbol{d}$ is a vector in $\mathscr{R}^3$.

A general framework to compute the PD is based on Minkowski sums. The Minkowski sum, $P \oplus Q$, is defined as a set of pairwise sums of vectors from P and Q. In other words, $P \oplus Q = \{\boldsymbol{p}+\boldsymbol{q} | \ \boldsymbol{p} \in P, \boldsymbol{q} \in Q\}$. Furthermore, $P \oplus -Q$ is defined by negating Q; i.e. $P \oplus -Q = \{\boldsymbol{p}-\boldsymbol{q} | \ \boldsymbol{p} \in P, \boldsymbol{q} \in Q\}$.

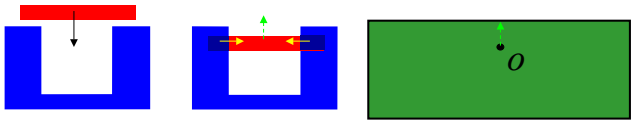
It is well known that one can reduce the problem of computing the PD between P and Q to a minimum distance query on the surface of their Minkowski sum , $P \oplus -Q$ [Cameron 1997]. More specifically, if two polyhedra P and Q intersect, then the difference vector, $\boldsymbol{O}_Q - \boldsymbol{O}_P$, between the origins[1] of P and Q is inside $P \oplus -Q$. Let us denote $\boldsymbol{O}_Q - \boldsymbol{O}_P$ by $\boldsymbol{O}_{Q-P}$. The $PD(P,Q)$ is defined as a minimum distance from $\boldsymbol{O}_{Q-P}$ to the surface of $P \oplus -Q$.

3.2 Local Vs. Global Computations

The computation of the PD between two polyhedral models is a global problem and it is rather difficult to localize it using some 'divide-and-conquer' approach. A local solution computed using intersecting features or boundaries may not be correct, as shown in Fig.1-(b). However, as Fig. 1-(c) shows, the minimum distance from the origin, $\boldsymbol{O}$, to the surface of the Minkowski sum of the two polygons in Fig. 1-(b) is the global PD. Furthermore, a local

[1]The origin of a polyhedron refers to the origin of the local coordinate system of the polyhedron with respect to the global coordinate system. Also, throughout the rest of the paper, the origin of the Minkowski sum will refer to the difference vector of the origins of two polyhedra.

solution to the PD problem may fail to compute a correct response for dynamic simulation, as explained in Section 7.



(a) (b) (c)

Figure 1: 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 arrows) based on $O(nm)$ intersections may not provide a global PD which is denoted as a dotted 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.*

3.3 Our Approach

It is relatively easier to compute Minkowski sums of convex polytopes as compared to general polyhedral models. However, for non-convex polyhedra in 3D, the Minkowski sum can have $O(n^6)$ worst-case complexity [Dobkin et al. 1993]. 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.

In order to overcome its combinatorial and computational complexity, we use a *surface-based* convex decomposition of the boundary and utilize the graphics rasterization hardware to estimate the PD. Moreover, we do not explicitly compute the boundary of the union or any approximation to it. Rather, we perform the *closest point query* using a multipass algorithm that computes 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 image-space resolution used for this computation.

We improve the performance of the basic algorithm highlighted above using a number of acceleration techniques. These include hierarchical representation based on convex bounding volumes, object-space culling approaches and image-space acceleration techniques applied to the multipass algorithm. These are explained in detail in Section 5.

The resulting algorithm includes a pre-computation phase as well as a runtime query. The pre-computation phase consists of the following steps:

1. Decompose the boundary of each polyhedron into convex patches using a greedy approach (Sec. 4.1).
2. Compute a bounding volume hierarchical representation of the model. Each node in the tree corresponds to a convex polytope and each leaf is a convex hull of a decomposed convex patch (Sec. 5.2).

Given two polyhedra and their bounding volume representations, the runtime phase of the algorithm proceeds in the following manner:

1. Compute an upper estimate to the amount of PD. Let that estimate be d_{est}. Initially we compute an estimate based on the root nodes of each tree (Sec. 5.3).
2. At each level of the two hierarchies, repeat the following steps:
   (a) Consider all pairwise combinations of nodes at the current level. Cull away all the pairs that are non-overlapping and are more than d_{est} apart (Sec. 5.3).
   (b) Compute pairwise Minkowski sums of the rest of the node pairs that have not been culled away (Sec. 4.2).
   (c) Perform the closest point query using the rasterization hardware to compute a PD estimate (Sec. 4.3).

The entire pipeline of our PD algorithm is illustrated in Fig. 2.

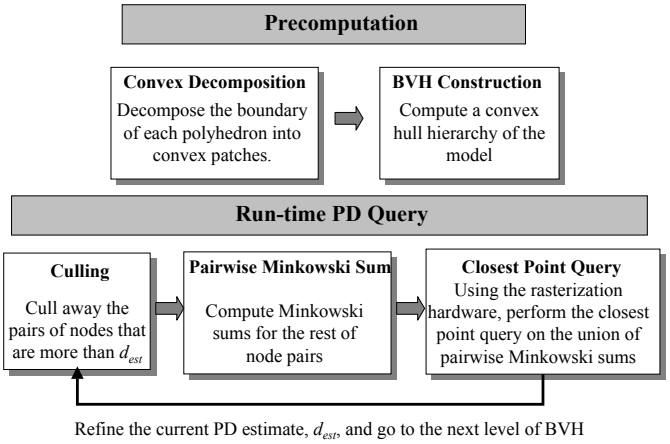


Figure 2: PD Computation Pipeline

3.4 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.

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

4 Penetration Depth Computation

In this section, we present our basic algorithm for global PD computation. It involves decomposing the boundary of the model into convex patches, computing their pairwise Minkowski sums, and then performing a closest point query using rasterization hardware.

Given two intersecting polyhedra, the PD query reports a PD scalar value and direction, along with the associated PD features[2].

[2]These are a pair of features that realize the reported PD. The PD value is the same as the inter-distance between planes which locally support the PD features.

In this case, the origin is contained inside the Minkowski sum of the two polyhedra. Fig. 3 illustrates the result computed by the PD query.

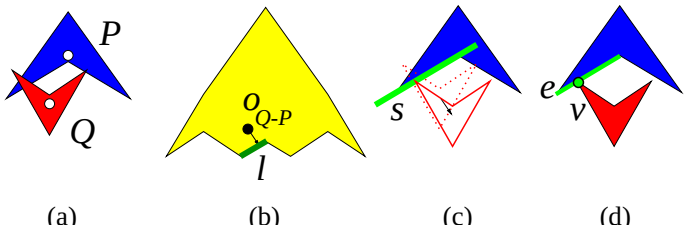


Figure 3: Simple Penetration Depth Computation in 2D. *(a) Two polygons P and Q intersect. (b) The Minkowski sum $P \oplus -Q$ is computed, and the minimum distance from the origin $\mathbf{O}_{Q-P}$ to $\partial(P \oplus -Q)$ is determined. (c) When P is translated by the amount of PD, there exists a line s locally supporting both polygons. In this case, the edge/vertex feature pair, (e,v), as shown in (d) makes up the PD features. The PD features are also identified in (b) as a corresponding line l.*

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 except for their shared edges, and the union of all the c_i's covers the entire boundary of P, ∂P.

We compute the convex patches, c_i's, by dualizing the polyhedral surface and performing a graph search on it in a greedy manner. First, we construct a dual graph G of the polyhedral surface ∂P by reversing the roles of faces (F) and vertices (V) in ∂P, while using the same edges (E) in G from ∂P [Chazelle et al. 1997; Ehmann and Lin 2001]. We traverse the dual graph G by adding faces into a current convex patch c_i as long as it maintains its convexity. We repeat this process until we cover the entire boundary of ∂P. For example, Fig. 7-(h) is a convex surface decomposition for a wrinkled torus model, Fig. 7-(a).

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 is completely contained in 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 3.3. 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}. We use a simple algorithm based on the convex hull property of Minkowski sums:

$$P \oplus Q = \mathbf{CH}(\{\boldsymbol{v}_i + \boldsymbol{v}_j | \boldsymbol{v}_i \in V_P, \boldsymbol{v}_j \in V_Q\}) \tag{2}$$

Here, **CH** denotes the convex hull operator, and V_P, V_Q represent the sets of vertices, respectively in polyhedra P and Q. Based on this fact, we compute the Minkowski sum as follows:

1. Compute the vector sum between all possible pairs of vertices from each polytope.
2. Compute their convex hull.

4.3 Closest Point Query Using Graphics Hardware

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

$$M = \bigcup_{ij} M_{ij}. \tag{3}$$

Our goal is to compute the closest point on the boundary of M, i.e. ∂M, from the origin. We use z-buffer polygon rasterization hardware to perform this query up to image-space resolution. The main idea is to visualize ∂M from the origin without computing a surface representation of ∂M explicitly. After that we compute the closest point, the distance and the direction.

4.3.1 Visualizing the Boundary of the Union

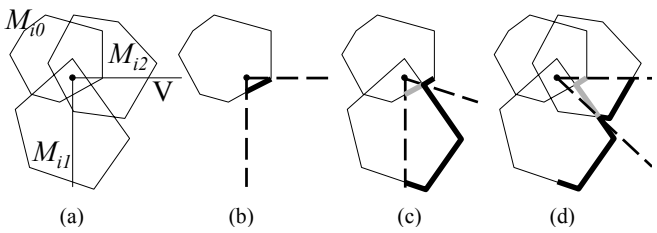


Figure 4: Visualizing the Boundary of the Union From Inside. *In (a), V is the current view-frustum. In (b), M_{i0} is rendered, and a new ∂M is constructed (thick line). In (c), when M_{i1} is rendered, it opens up a new window (dotted line), and the update region (thick gray line) on the current ∂M is established. Thus a new ∂M (thick line) is constructed. In (d), we perform the same procedure for M_{i2}.*

Our algorithm for visualizing ∂M from a point inside is essentially a ray-shooting procedure from the origin to ∂M, and incrementally expands the front of ∂M. For example, in Fig. 4, we expand the current ∂M (thick line) by repeatedly rendering M_{i0}, M_{i1}, M_{i2}. Each time M_{i0}, M_{i1}, M_{i2} are rendered, as shown in Fig. 4(b)-(d), it opens up a new window (shown as dotted line) of the update region (thick gray line) on the current ∂M.

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 rendering 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 render the backfaces of M_{ij} into the opened window using the maximum depth test. However, we should not render the backfaces of M_{ij}, that are created by non-original (virtual) faces of P and Q. In other words, we should allow the ray to hit only ∂M.

In summary, the basic algorithm simply performs the following procedure:

1. Initialize ∂M^0 to infinity.
2. Repeat steps 3-5 m times
3. Repeat steps 4-5 for each M_{ij}
4. Render front faces of M_{ij}, and using the standard stencil operation, open a window where the depth value of the front faces is less than that of the current ∂M^k.
5. Classify the backfaces of M_{ij} into original and non-original. Render only the original back faces of M_{ij} where the depth value of the back faces is greater than that of the window. This updates the ∂M^k in the window.

After the mth iteration in step 2 highlighted above, the algorithm correctly finds the portion of ∂M that is visible from the origin in

the following sense. After the kth iteration in step 2, ∂M^k includes the subset of ∂M that the ray can reach with less than or equal to $k-1$ *hops* from the origin. Here, the *hop* on some point $\boldsymbol{p}$ on ∂M means the number of M_{ij}'s the ray should pass through to reach $\boldsymbol{p}$. For example, ∂M^1 includes the possible contribution to the final ∂M of all M_{ij}'s that contain the origin and have zero hops. Therefore, by induction on k, we correctly find the portion of ∂M that is visible from the origin after the mth iteration.

4.3.2 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 reads back the Z-buffer to obtain the depth values for each pixel. However, these depth values have undergone the perspective depth transformation and do not contain the non-linearity that is present in the distance values.

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. This depth transformation is CPU-bound, and this operation typically takes a few milliseconds.

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.

5 Acceleration Techniques

The global PD computation algorithm described in Section 4 estimates 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 the worst case, the convex decomposition algorithm can result in $O(n)$ patches and this can lead to $O(n^2)$ pairwise Minkowski sums, M_{ij}. Furthermore, the cost of the closest point query using rasterization hardware can be as high as $O(m^2)$, where m is the number of convex polytopes. This results in $O(n^4)$ worst case complexity for the PD estimation algorithm.

In this section, we present a number of acceleration techniques to improve its performance. These include hierarchical culling and image-space acceleration techniques.

5.1 Object Space 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 5.1 *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 $\partial(M - M_{ij})$.*

For example, in Fig. 5, there are two intersecting polygons P and Q. We estimate d_{est} based on the convex hull of P and Q (Fig. 5-(b)). Then, we can cull away the pairs whose separation distance is more than d_{est} (Fig. 5-(c)).

Based on the Lemma 5.1, we can cull away all pairs of convex polytopes, C_i^P and C_j^Q, whose separation distances are more than

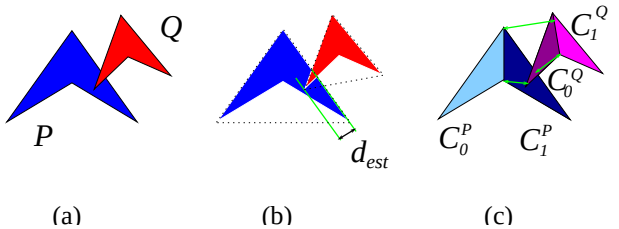


Figure 5: Object Space Culling. *(a) There are two intersecting polygons P (decomposed into C_0^P, C_1^P) and Q (decomposed into C_0^Q and C_1^Q). (b) Based on the convex hull of P and Q, we first estimate the PD as d_{est}. (c) Using d_{est}, we can cull away pairs (C_0^P, C_0^Q), (C_0^P, C_1^Q), (C_1^P, C_1^Q), whose separation distances are more than d_{est}.*

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 [Lin and Canny 1991; Cameron 1997]. 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. For example, Fig. 7-(b) $\sim$ (h) shows a BV hierarchy for the torus model, Fig. 7-(a).

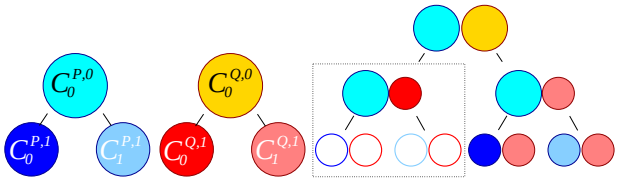
This hierarchy is used in our runtime algorithm to speed up the intersection and separation distance queries for the culling algorithm. Furthermore, each level of the hierarchy provides an approximation of the model, which is used by the PD estimation algorithm.

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 [Cameron 1997; Bergen 2001; Kim et al. 2002a] 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 of 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.



(a) Two BV hierarchies (b) BV hierarchy Traversal

Figure 6: Hierarchical Culling. *(a) shows two BV hierarchies for two different objects. (b) shows a snapshot of the traversal on the BV hierarchies. During the traversal, it turns out that node $C_0^{P,0}$ and node $C_0^{Q,1}$ are non-overlapping and their inter-distance is greater than a upper bound on the current PD estimation. Thus, no more traversal is performed between the children nodes of $C_0^{P,0}$ and $C_0^{Q,1}$.*

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}^h as the estimated PD between P and Q.

Fig. 6-(a) shows BV hierarchies for two different objects P and Q, and Fig. 6-(b) shows a snapshot of how the BV hierarchy traversal is performed. Without the culling scheme, one should consider all four pairs between $C_0^{P,1}$, $C_1^{P,1}$ and $C_0^{Q,1}$, $C_1^{Q,1}$. However, during the traversal, $C_0^{P,0}$ and $C_0^{Q,1}$ are found to be non-overlapping and they are more than d_{est} apart. In this case, no more traversal is needed between the children nodes of $C_0^{P,0}$ and $C_0^{Q,1}$.

5.4 Image Space 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 4.3.1. All the pairwise Minkowski sums in this subset have a zero *hop*. We identify this subset, say l out of a total of m pairs of M_{ij}'s, by checking whether the corresponding convex polytopes, C_i^P and C_j^Q, overlap [Lin and Canny 1991; Cameron 1997; Ehmann and Lin 2001]. 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 5.1. These distances are also computed in object space. Finally, for each view, 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 object-space view frustum culling significantly reduces the number of primitives rendered during each iteration of the algorithm.

6 Implementation and Results

In this section, we describe the implementation of our PD computation algorithm and demonstrate its performance on different benchmarks. For extensive experiment results on our PD computation and its optimization, we refer the readers to see [Kim et al. 2002b]

6.1 Implementation Issues

Our algorithm requires quick and robust implementations of the separation distance query between convex polytopes and convex hull computation in 3D.

We use the SWIFT++ implementation of the *Voronoi marching* technique [Ehmann and Lin 2001] 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 [Barber et al. 1993] for the pairwise Minkowski sums M_{ij}'s, by using the convex hull based algorithm described in Section 4.2.

We used the OpenGL graphics library to implement the closest point query. 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 reasonable accuracy for our applications, and they also balance the computation time between the object space and the image space algorithms.

6.2 Performance Benchmarking

We have applied our PD algorithm to four benchmarks: interlocked tori, touching tori, interlocked "grates" and a pair of alphabet models, with their relative configuration shown in Fig. 8.

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"). Detailed timings for some levels of the hierarchy are given in Table 2.

We also observe that the performance of our algorithm depends heavily on the extent of object-space culling, which is directly related to the amount of inter-penetration between the objects. Therefore, for applications that have spatial and temporal coherence between successive instances, our algorithm performs quite well since penetration is typically shallow during successive time steps. As a result, the algorithm is able to cull away a very high percentage of Minkowski pairs (as shown in Table 2) and is quite fast in practice.

6.3 Performance Speedup by Acceleration Techniques

In Table 3, we also compare 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

6.4 Accuracy of PD Computation

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 tightness of the upper bound varies based on the underlying precision of the object-space and image-space computations. The accuracy of the algorithms for surface decomposition, Minkowski sum computations and object-space culling is governed by the precision of floating-point CPU-based hardware, which typically has 53 bits of mantissa. However, the rasterization errors and precision of image-space computations governs the tightness of the resulting answer. The main sources of these errors are:

1. The discretization of ray directions to lie on a pixel grid for each view.

2. The fixed precision of the Z-buffer.

Increasing the resolution of the grid decreases the worst-case angular error that is proportional to the distance between adjacent pixels. Moreover, constructing tighter bounds on the minimum and maximum distances in each view (near and far plane distances), decreases the Z-buffer precision error.

7 Application to Rigid Body Simulation

In this section, we describe the application of our PD algorithm to dynamic simulation of rigid bodies, as shown in Fig. 9. This includes contact response computation in penalty-based methods [McKenna and Zeltzer 1990; Moore and Wilhelms 1988], as well as a time stepping technique for either impulse-based [Mirtich and Canny 1995] or constraint-based simulation [Baraff 1992; Baraff 1994; Witkin and Baraff 1997].

PD computation is important for robust physically-based animation. Inter-penetration is often unavoidable in numerical simulations, unlike in the physical world. Moreover, for applications involving articulated joints, stacking objects and parts assembly, the bodies are nearly in contact or actually touching each other all the time. Our PD computation algorithm provides a consistent and accurate measure of PD for rigid body dynamic simulation using various (including penalty-based, constraint-based, or impulse-based) simulation methods.

7.1 Penalty-based Methods

In penalty-based methods, the forces between rigid bodies are proportional to the amount of inter-penetration. Let d be the translational PD, $\boldsymbol{n}$ the direction of penetration and k a stiffness constant. The force vector $\boldsymbol{F}$ is given as:

$$\boldsymbol{F} = (k \cdot d)\boldsymbol{n} \tag{4}$$

Local vs. Global PD Computation: Localized approaches for computing the PD may fail to give a correct response for penalty-based methods. For example, in the configurations shown in Fig. 1, which illustrated 2D geometric models of letters 'C' and 'I', the forces based on localized values of PD may not prevent the rigid bodies from inter-penetrating. However, our global PD computation algorithm can provide a correct and robust response for such configurations. In the scenario shown in Fig. 10 several digits are interlocked. In this simulation, local approximations to the PD fail to report consistent or correct outputs for many pairs of digits.

7.2 Time Stepping

Unlike penalty-based methods that allow the models to inter-penetrate, some other simulation approaches impose strict non-penetration constraints. These include constraint-based simulation that distinguishes between resting contacts and colliding contacts. The resting contacts are classified based on the fact that the relative velocity is lower than a certain value. For colliding contacts, the impulsive forces are applied to prevent inter-penetration and preserve momentum properties. These impulsive forces are computed at the time of contact between the rigid bodies. In practice, the exact time of impact cannot be computed using analytic techniques. As a result, time stepping techniques are often used to estimate the time of collision [Stewart and Trinkle 1996; Mirtich 2000].

Some of the commonly used high-level scheduling algorithms for impulse-based or constraint-based rigid body simulation include retroactive detection and conservative advancement [Mirtich 2000]. Both of the techniques advance or retract the simulation time based on the separation distance between the objects and use some form of root finding algorithm to estimate the time of collision.

- **Bisection search.** Performing a simple bisection search in time is one of the commonly used technique to estimate the time of collision. This approach converges given a sufficient number of iterations. Moreover, it does not suffer from the inaccuracy of penetration depth estimation or approximations to the motion. However, it can take a long time to converge, if the time step used in the simulation is large.

- **Extrapolation.** If no information is gathered on the extent of penetration at the end of an interval, a common approach to predict the time of collision is based on extrapolation. The separation distance and the velocity of the closest features at the beginning of the interval are used for extrapolation. The

main problem with this approach arises when the two objects are penetrating at the estimated time of collision. In such cases, the predicted time of collision cannot be corrected.

- **Interpolation.** If penetration information between the two objects is known, it is possible to perform interpolation. Unlike extrapolation, we make use of penetration information at the end of the interval and an iterative interpolatory scheme can be used to estimate the time of collision [Witkin and Baraff 1997]. As shown in Fig. 1, localized estimations can deviate largely from the actual penetration depth. That can result in inaccurate estimation of the time of contact. A global and exact computation of penetration depth provides a faster and more robust convergence of the root finding scheme. Depending on the position and velocity information, the algorithm can select an appropriate order of interpolation. For example, [Mirtich 1998] performs linear interpolation using the separation distance at the beginning of the interval and an estimated penetration depth at the end of the interval. If we also know the velocity of the object, we can perform higher order interpolation. If only the initial velocity and the scalar value of penetration distance are known, but not the penetration direction or penetrating features, we can use a quadratic interpolation scheme. In such cases, the interpolation is performed based on the initial separation distance, and velocity, as well as the penetration depth at the end of the interval. If the algorithm also knows the penetrating features, then we can compute the relative velocity and perform cubic interpolation.

In our implementation of the time stepping scheme for dynamic simulation, we utilize the knowledge about PD features and direction and use a cubic interpolation scheme to estimate the time of collision. In general, using higher order interpolation allows us to take large steps in the simulation [Witkin and Baraff 1997]. However, sometimes it is not possible to take large time steps in the simulation because of the following reasons:

- The stability of numerical integration.
- The frequency of collision events. Even if the system is numerically stable, a high frequency of contacts between the objects make the effective time step small. In such cases, the time stepping can not benefit from higher order interpolation.

Given two inter-penetrating objects, our PD algorithm computes the penetration depth d, the direction $\boldsymbol{n}$ and the penetrating features (as shown in Fig. 3). Object's motion in the last time step is approximated as a one-dimensional motion by projecting it onto the penetration direction. We compute a cubic interpolation of the motion, using the separation distance s and the relative velocity of the closest features at the beginning of the interval $\boldsymbol{v_s}$, along with the penetration depth and the relative velocity of the penetration features at the end of the interval $\boldsymbol{v_d}$. The cubic function is expressed as:

$$x(t) = At^3 + Bt^2 + Ct + D.$$

We treat $x(t)$ as the one dimensional distance function between the closest features or penetrating features of the rigid bodies. The parameters A, B, C and D are generic constants of a cubic polynomial that are computed by solving the following set of linear equations:

$$\begin{aligned} x(0) &= s = D, \\ x(T) &= d = AT^3 + BT^2 + CT + D, \\ \dot{x}(0) &= \boldsymbol{v_s} \cdot \boldsymbol{n} = C, \\ \dot{x}(T) &= \boldsymbol{v_d} \cdot \boldsymbol{n} = 3AT^2 + 2BT + C, \end{aligned}$$

where T is the size of the time step and $\boldsymbol{n}$ is the direction of penetration. After computing the coefficients of the cubic polynomial, we compute its first real root in the interval $[0, T]$.

Based on this approach, our algorithm is able to compute the time of collision using fewer iterations, as compared to earlier methods. If there is any error, e.g. the cubic polynomial has no real root in the interval $[0, T]$, then we use bisection search to estimate the time of collision.

In one of the example scenarios, as shown in the video at *http://gamma.cs.unc.edu/PD* and in Fig. 9, geometric models of 200 letters and digits fall along a structure consisted of multiple funnels and ramps. The letters and digits have an average complexity of 250 triangles in their boundary surface, which is decomposed into roughly 60 convex pieces. Each frame (at 30fps) of the dynamic simulation for this complex scenario takes about two minutes to compute.

For the second scenario in the video at the same web site and in Fig. 10, there are 10 digit models dropped into a bowl. The geometric model of each digit has an average complexity of 250 triangles. The bowl consists of 176 triangles and its surface is decomposed into roughly 65 convex pieces. Some of the digits come into an interlocking position as they fall into the bottom of the bowl. These are challenging scenarios for contact computation and response. Each frame of this simulation takes about 18 seconds to compute.

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 provides an accurate and robust measure of penetration between overlapping objects for physically-based animations. We have used it for contact response computation and designed a time-stepping method for rigid body dynamic simulation.

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. We would also like to further compare the performance, in terms of both running time and accuracy, of our PD-based time-stepping algorithm with earlier approaches. Currently our PD algorithm only computes the minimum translational distance to separate two overlapping objects. It will 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, Intel and a fellowship of the Government of the Basque Country.

References

AGARWAL, P., GUIBAS, L. J., HAR-PELED, S., RABINOVITCH, A., AND SHARIR, M. 2000. Penetration depth of two convex polytopes in 3D. *Nordic J. Computing 7*, 227–240.

BARAFF, D. 1992. *Dynamic simulation of non-penetrating rigid body simulation*. PhD thesis, Cornell University.

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

BARBER, B., DOBKIN, D., AND HUHDANPAA, H. 1993. The quickhull algorithm for convex hull. Tech. Rep. GCG53, The Geometry Center, MN.

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

CAMERON, S., AND CULLEY, R. K. 1986. Determining the minimum translational distance between two convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, 591–596.

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

CHAZELLE, B., DOBKIN, D., SHOURABOURA, N., AND TAL, A. 1997. Strategies for polyhedral surface decomposition: An experimental study. *Comput. Geom. Theory Appl. 7*, 327–342.

DOBKIN, D., HERSHBERGER, J., KIRKPATRICK, D., AND SURI, S. 1993. Computing the intersection-depth of polyhedra. *Algorithmica 9*, 518–533.

EHMANN, S., AND LIN, M. C. 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001) 20*, 3.

EPSTEIN, D., JANSEN, F., AND ROSSIGNAC, J. 1989. Z-buffering rendering from CSG: The trickle algorithm. Tech. rep., IBM Research Report RC15182.

FISHER, S., AND LIN, M. C. 2001. Deformed distance fields for simulation of non-penetrating flexible bodies. *Proc. of EG Workshop on Computer Animation and Simulation*.

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

GOLDFEATHER, J., HULTQUIST, J. P. M., AND FUCHS, H. 1986. Fast constructive-solid geometry display in the Pixel-Powers graphics system. In *Proc. of ACM SIGGRAPH*, vol. 20, 107–116.

GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, 171–180.

GREGORY, A., MASCARENHAS, A., EHMANN, S., LIN, M. C., AND MANOCHA, D. 2000. 6-DOF haptic display of polygonal models. *Proc. of IEEE Visualization Conference*.

GUIBAS, L., AND SEIDEL, R. 1987. Computing convolutions by reciprocal search. *Discrete Comput. Geom 2*, 175–193.

HOFF, K., CULVER, T., KEYSER, J., LIN, M., AND MANOCHA, D. 1999. Fast computation of generalized Voronoi diagrams using graphics hardw are. *Proceedings of ACM SIGGRAPH*, 277–286.

HOFF, K., ZAFERAKIS, A., LIN, M., AND MANOCHA, D. 2001. Fast and simple geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics*.

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

KAUL, A., AND ROSSIGNAC, J. 1992. Solid-interpolating deformations: construction and animation of pips. *Computer and Graphics 16*, 107–116.

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

KIM, Y. J., OTADUY, M. A., LIN, M. C., AND MANOCHA, D. 2002. Fast penetration depth computation using rasterization hardware and hierarchical refinement. Tech. rep. TR02-014, UNC-Chapel Hill.

KLOSOWSKI, J., HELD, M., MITCHELL, J. S. B., ZIKAN, K., AND SOWIZRAL, H. 1998. Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Trans. Visualizat. Comput. Graph. 4*, 1, 21–36.

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

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

MCNEELY, W., PUTERBAUGH, K., AND TROY, J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. *Proc. of ACM SIGGRAPH*, 401–408.

MIRTICH, B., AND CANNY, J. 1995. Impulse-based simulation of rigid bodies. In *Proc. of ACM Interactive 3D Graphics*.

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

MIRTICH, B. 2000. Timewarp rigid body simulation. *Proc. of ACM SIGGRAPH*.

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

ONG, C. J., AND GILBERT, E. 1996. Growth distances: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation 12*, 6.

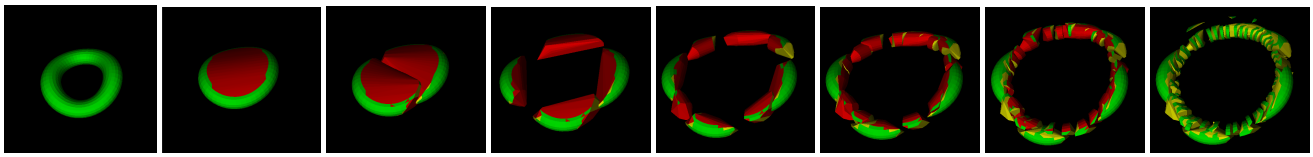
ROSSIGNAC, J., MEGAHED, A., AND SCHNEIDER, B.-O. 1992. Interactive inspection of solids: Cross-sections and interferences. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, E. E. Catmull, Ed., vol. 26, 353–360.

STEWART, D. E., AND TRINKLE, J. C. 1996. 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.

THEOHARIS, T., PAPAIANNOU, G., AND KARABASSI, E. 2001. The magic of the Z-buffer: A survey. *Proc. of 9th International Conference on Computer Graphics, Visualization and Computer Vision, WSCG*.

WIEGAND, T. F. 1996. Interactive rendering of CSG models. *Computer Graphics Forum 15*, 4, 249–261.

WITKIN, A., AND BARAFF, D. 1997. *Physically Based Modeling: Principles and Practice*. ACM Press. Course Notes of ACM SIGGRAPH.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 7: Convex Surface Decomposition and Bounding Volume Hierarchy. *(a) shows an original model for a torus, and (h) shows its convex surface decomposition. From (b) to (h), the figure shows a BV hierarchy of the torus from root level to leaf level. In the figure, the green color indicates an original face in the model, the red color highlights a virtual face created by convex hull computation, and the yellow color indicates a virtual face created while converting a convex patch to a convex piece.*

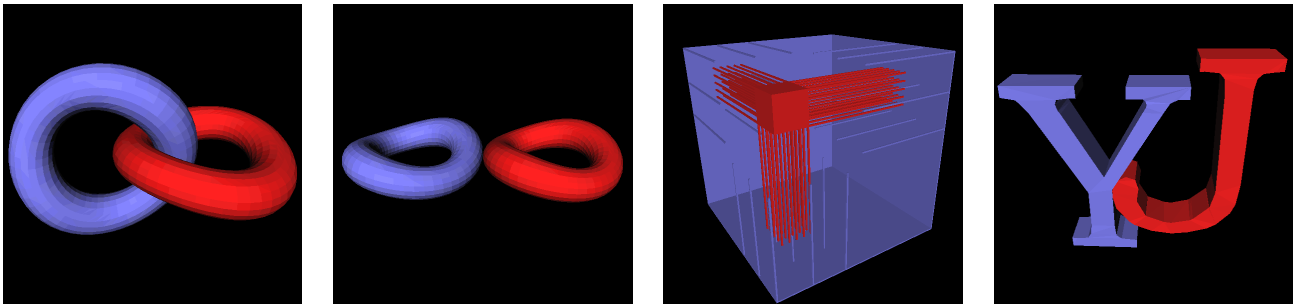


Figure 8: PD Benchmark Models. *From left to right: interlocked tori, touching tori, interlocked grates, and letters.*

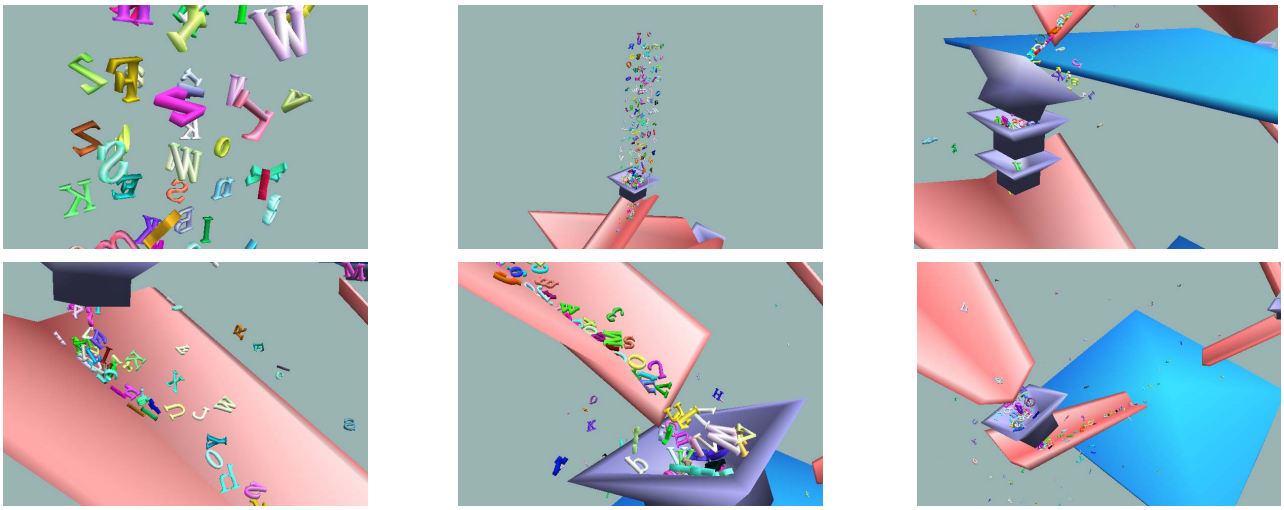


Figure 9: Application to Rigid-Body Dynamic Simulation. *Our algorithm is used to perform smarter time stepping in a dynamic simulation. A sequence of snapshots (from left to right, top to bottom) are taken from a rigid-body simulation of* 200 *models of letters and numerical digits falling onto a structure consisting of multiple ramps and funnels.*

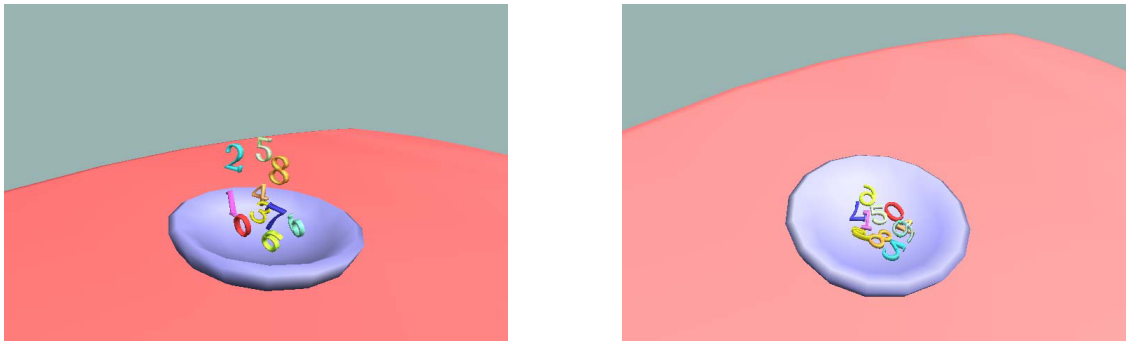


Figure 10: Challenging scenario with interlocked digits. *On the left,* 10 *digits are falling onto a bowl. On the right, a resting position of these digits is shown. Localized approaches to compute PD fail for some pairwise digits in this interlocked configuration.*