PolyDepth: Real-time Penetration Depth Computation using Iterative Contact-Space Projection

Changsoo Je[1,2], Min Tang[1], Youngeun Lee[1]
Minkyoung Lee[1] and Young J. Kim[1]

[1]Ewha Womans University, Seoul, Korea
[2]Korea Institute of Patent Information, Seoul, Korea

July 22, 2010

Abstract

We present a real-time algorithm that finds the penetration depth (PD) between general polygonal models based on iterative and local optimization techniques. Given an in-collision configuration of an object in configuration space, we find an initial collision-free configuration. We project this configuration on to a local contact space using a variant of continuous collision detection algorithm and construct a linear convex cone around the projected configuration. We then formulate a new projection of the in-collision configuration on to the convex cone as a linear complementarity problem (LCP), which we solve using a type of Gauss-Seidel iterative algorithm. We repeat this procedure until a locally optimal PD is obtained. Our algorithm can process complicated models consisting of tens of thousands triangles at interactive rates.

1 Introduction

Measuring the distance between geometric objects is an important problem in computer graphics, virtual reality, geometric modeling, computational geometry, CAD/CAM and robotics [LM03]. When objects are disjoint, the Euclidean distance between their closest points, also known as the separation distance, is an obvious measure of distance. However, when objects overlap, the separation distance is undefined and a different measure is needed to quantify the amount of interpenetration. Different penetration measures have been introduced, including penetration depth [CC86], generalized penetration depth [ZKVM07], pointwise penetration depth [TLK09], growth distance [Ong93], and penetration volume [WZ09].

Penetration depth (PD) has been widely used by the computational geometry community. It is the distance that corresponds to the shortest translation required to separate two overlapping, rigid objects [DHKS93]. Many applications can benefit from PD computations. In physically-based animation, the PD can be used to locate the point of application for impulses [GBF03], or used to find the time of contact in time-stepping methods [KOLM02a]. In virtual prototyping, the PD can be used to verify tolerances [KOLM02b]. The length of a PD can be used to determine the appropriate force feedback in six-degree-of-freedom haptic systems [KOLM03]. Retraction-based motion planning algorithms can use PD to find a collision-free sample in the narrow passage amongst obstacles [ZM08], and PDs can be used to determine the existence of a path in planning scenarios [ZKM08].

However, it is well-known that much more computation is required to determine a PD than a separation distance. For general polyhedral objects, with a total of n faces, the computation time is $O(n^6)$ [KLM04]. This has motivated some researchers to find a more tractable approximation of the PD [KOLM02b, KOLM02a, RL06]. Unfortunately, these methods are either too slow for interactive applications [KOLM02b, KOLM02a] or provide no guaranteed error bound [RL06].

Main Contributions: We present a real-time algorithm to approximate the PD between arbitrary, polygon-soup models. Our algorithm actually determines an upper bound on the PD and we show empirically that this is a tight bound on the exact value obtained from Minkowski sums. We also show how to obtain a set of *local* PD values from the PD solution. These local PDs characterize the amount of local overlap in each of several interpenetrating regions.

Our algorithm is based on iterative and local optimization techniques. Given an in-collision configuration for an object, we find an initial collision-free configuration in configuration space. Then, we project this initial configuration on to a contact space using a variant of a continuous collision detection algorithm and construct a linear convex cone around the projected configuration. We then formulate the projection of the in-collision configuration onto this cone as a linear complementarity problem (LCP), which we solve using a Gauss-Seidel iteration. This runs until a locally optimal solution is obtained.

Because ours is an iterative algorithm, finding a good initial configuration is crucial. We therefore propose several search techniques to find the configuration from geometric properties such as the distance between centroids, maximally clear configuration, sampling-based search or random search, as well as exploiting application-dependent information such as motion coherence.

We have implemented our algorithm, *PolyDepth*, and benchmarked its performance in various complicated scenarios, such as random and pre-calculated configurations, and configurations governed by rigid-body dynamics. In all these scenarios, our algorithm achieves a highly interactive performance while providing a tight estimate of the PD value.

Organization: The rest of the paper is organized as follows. In Sec.2, we briefly survey topics relevant to PD computations. We provide some preliminary information needed to understand our algorithm in Sec. 3, and give an overview of the algorithm. In Secs. 4 and 6 we explain the two central techniques of our algorithm, out- and in-projection as well as local optimization techniques. In Sec. 5 we explain how we find a good initial collision-free configuration. We present our experimental results and analyze the performance of PolyDepth in different benchmarks in Sec. 7, and conclude the paper in Sec. 8.

2 Previous Work

There are several different types of PD algorithms for different model geometries and objectives.

Convex Polytopes: a straightforward algorithm was presented to compute the PD between two convex polytopes by computing their Minkowski sums [CC86]. If the direction of motion is known, a minimal translational motion in this direction can be found using a multi-resolution mesh hierarchy [DHKS93]. A randomized algorithm was presented by [AGHp$^+$00]. These algorithms provide exact solutions, but they are difficult to implement; in fact, no good implementations are known. However, various approximate algorithms have been developed based on upper and lower bounds on the PD [Cam97], expansion of polyhedral approximations [Ber01], and dual-space expansion

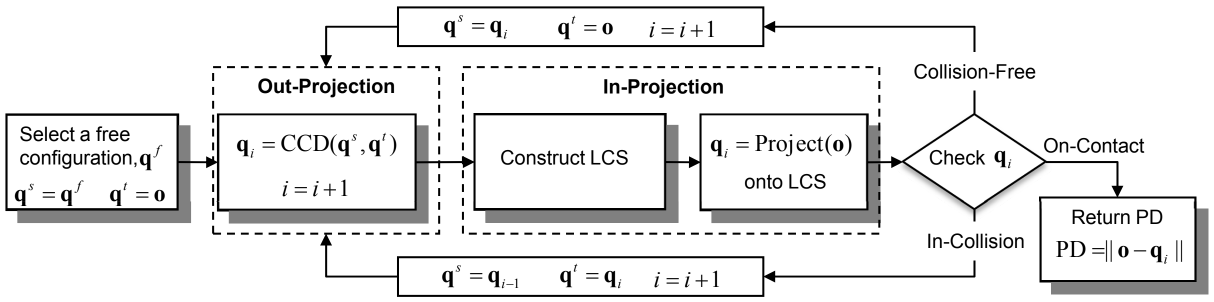

$\mathbf{q}^s = \mathbf{q}_i \quad \mathbf{q}^t = \mathbf{o} \quad i = i+1$
Out-Projection
In-Projection
Collision-Free
Select a free configuration, $\mathbf{q}^f$
$\mathbf{q}^s = \mathbf{q}^f \quad \mathbf{q}^t = \mathbf{o}$
$\mathbf{q}_i = \mathrm{CCD}(\mathbf{q}^s, \mathbf{q}^t)$
$i = i+1$
Construct LCS
$\mathbf{q}_i = \mathrm{Project}(\mathbf{o})$ onto LCS
Check $\mathbf{q}_i$
On-Contact
Return PD
$\mathrm{PD} = \| \mathbf{o} - \mathbf{q}_i \|$
In-Collision
$\mathbf{q}^s = \mathbf{q}_{i-1} \quad \mathbf{q}^t = \mathbf{q}_i \quad i = i+1$

Figure 1: **PD computation pipeline.**

[KLM04]. The convex PD problem requires only $O(n^2)$ time in the worst case, where n is the total number of faces in the polytope.

Non-convex Polyhedra: the computational complexity of PD for non-convex polyhedra is $O(n^6)$, and thus no practical exact algorithms exist. A hierarchical refinement technique combined with GPU-assisted ray-shooting can provide an upper bound on the PD [KOLM02a]. Redon and Lin [RL06] also proposed a CPU/GPU hybrid method of computing a lower bound on the PD, and this algorithm is applicable to polygon-soup models. And Lien [Lie08a, Lie08b] presented a sampling-based approach to PD computation based on approximate Minkowski sum. More recently, Hachenberger [Hac09] presented a method of obtaining an exact Minkowski sum based on convex decomposition. However, these approaches are rather slow, and some require an explicit boundary representation of Minkowski sums that has to be re-evaluated whenever the orientation of an object is changed.

A somewhat different route is to use distance fields that has the advantage of being applicable to deformable models [FL01]. Moreover, a GPU can be used to accelerate the computation of distance fields [HZLM02, SGG$^+$06]. However, these approaches only provide a lower bound on a PD. There is also a variant of PD called a pointwise PD which is defined as the distance between the points of deepest interpenetration between two objects. Pointwise PDs can be found using Hausdorff distance [TLK09]. However, a pointwise PD is merely a lower bound on the PD. A lower bound on a PD cannot be used to achieve separation between overlapping objects, but an upper bound can.

Generalized Penetration Depth: the constraints of some applications mean that a translation is not sufficient to separate intersecting objects, and thus does not provide a useful measure of inter-penetration. In these cases, more complicated motions need to be considered. Zhang et al. proposed a generalized penetration depth, which is the minimal combination of translational and rotational motion needed to separate two objects [ZKVM07, ZKM07]. More recently, kinematical geometry has been used to compute generalized PDs [NPR07]. Non-Euclidean distance, such as growth distance [OG96, Ong93], can also be used as a measure of inter-penetration . Finally, some researchers have investigated penetration volumes instead of PD [WZ09], but the relation between this and the PD is questionable.

3 Preliminaries

We will now define the problem of penetration depth computation between general polygonal models and give an overview of our approach.

3.1 Problem Formulation

Suppose we have two objects $\mathfrak{A}$ and $\mathfrak{B}$ in $\mathbb{R}^3$. Without loss of generality, we will assume that $\mathfrak{A}$ is movable by translation and $\mathfrak{B}$ is stationary, and both objects have the common global origin $\mathbf{o}$. If $\mathfrak{A}$ and $\mathfrak{B}$ are polyhedral objects and are interpenetrated, their penetration depth $\mathrm{PD}(\mathfrak{A}, \mathfrak{B})$ is formally defined as [DHKS93]:

$$\mathrm{PD}(\mathfrak{A}, \mathfrak{B}) = \min\{\|\mathbf{d}\| \mid \mathrm{interior}(\mathfrak{A} + \mathbf{d}) \cap \mathfrak{B} = \emptyset, \forall \mathbf{d} \in \mathbb{R}^3\}. \tag{1}$$

It is well known that the PD computation is closely related to the Minkowski sum. Formally, the Minkowski sums $\mathfrak{A} \oplus \mathfrak{B}, \mathfrak{A} \oplus -\mathfrak{B}$ between two compact sets, $\mathfrak{A}$ and $\mathfrak{B}$, are defined [Ben66, Cam97] as follows:

$$\mathfrak{A} \oplus \mathfrak{B} = \{\mathbf{a} + \mathbf{b} | \mathbf{a} \in \mathfrak{A}, \mathbf{b} \in \mathfrak{B}\} \tag{2}$$

$$\mathfrak{A} \oplus -\mathfrak{B} = \{\mathbf{a} - \mathbf{b} | \mathbf{a} \in \mathfrak{A}, \mathbf{b} \in \mathfrak{B}\}. \tag{3}$$

We can then reduce the problem of computing the PD between $\mathfrak{A}$ and $\mathfrak{B}$ expressed by Eq. 1 to the search for the minimum distance between $\mathbf{o}$ and the boundary surface of their Minkowski sum, $\partial(\mathfrak{A} \oplus -\mathfrak{B})$ [DHKS93]. However, since the interior of the polygon-soup models that we wish to consider can be empty sets in $\mathbb{R}^3$, we need to define our PD problem in another way.

Our definition of PD still follows the intuitive notion of using a minimal translation to separate two overlapping models, even though the inside and outside of polygon-soup models is not properly defined. Thus, we define the PD to be the minimum distance from $\mathbf{o}$ to the boundary of the Minkowski sum $\mathfrak{A} \oplus -\mathfrak{B}$:

$$\mathrm{PD}(\mathfrak{A}, \mathfrak{B}) = \min\{\|\mathbf{d}\| \mid \mathrm{interior}(\mathfrak{A} \oplus -\mathfrak{B}) \cap \{\mathbf{o} + \mathbf{d}\} = \emptyset, \forall \mathbf{d} \in \mathbb{R}^3\}. \tag{4}$$

Essentially, the boundary of the Minkowski sum constitutes the *translational* contact space between $\mathfrak{A}$ and $\mathfrak{B}$, given that $\mathfrak{A}$ has a fixed orientation because it can only undergo translational motion. If $\mathfrak{A}$ and $\mathfrak{B}$ are polyhedra, then Eq.1 is equivalent to Eq.4.

3.2 Overview

Although the exact PD can be computed from Minkowski sums, the explicit computation of Minkowski sums between complicated polygon-soup models is too slow for interactive applications. Looking at Eq. 4, we see that finding the PD boils down to the search for the closest point to the origin on the contact space. Our algorithm approximates the contact space, which is the Minkowski sum boundary, only as far as is necessary to locate the closest point, refining its location until a locally optimal solution is obtained.

The main computational components of our iterative algorithm are two projections: (a) a projection from an in-collision configuration on to the contact space (the *in-projection*); and (b) a projection from a collision-free or contact configuration on to the contact space (the *out-projection*). These two projections allow us to obtain a contact configuration from an in-collision configuration or from a collision-free configuration.

The in-projection itself consists of two steps: (1) the construction of a local contact space (LCS) based on a linear complementarity problem (LCP) formulation; and (2) projection on to the LCS using a form of Gauss-Seidel iterative solver. The out-projection step is implemented using translational continuous collision detection (CCD). Both out- and in-projections are explained in detail in Secs. 4 and 6. The overall flow of our iterative algorithm is as follows (see Fig. 1):

1. Given two overlapping polygon-soup models $\mathfrak{A}$ and $\mathfrak{B}$, their common origin $\mathbf{o}$ should be inside the Minkowski sums $\mathfrak{A} \oplus -\mathfrak{B}$. We select a collision-free configuration $\mathbf{q}^f$ in the configuration space (Sec. 4.2). Then we perform out-projection from a source configuration $\mathbf{q}^s \equiv \mathbf{q}^f$ to a target configuration $\mathbf{q}^t \equiv \mathbf{o}$ using CCD (Sec. 4.1). We call the configuration projected on to the contact space the current configuration $\mathbf{q}_i$.
2. We then find the contact features of $\mathbf{q}_i$, such as vertex/face (VF) or edge/edge (EE) contacts. From these features, we construct a local contact space (LCS) around $\mathbf{q}_i$, which is a linear convex cone in configuration space.
3. We perform in-projection from $\mathbf{o}$ to the LCS by formulating this problem as an LCP, which we solve using a form of Gauss-Seidel algorithm (Sec. 5). Thus this in-projected configuration becomes the new current projection $\mathbf{q}_i$, and $\mathbf{q}_{i-1}$ is set to the configuration obtained from the previous out-projection.
4. Since $\mathbf{q}_i$ was obtained from the LCS, and not from the global contact space, $\mathbf{q}_i$ can be in-collision, in-contact or collision-free. We further classify the collision status of $\mathbf{q}_i$ by performing a static collision query on $\mathbf{q}_i$ as follows:
   (a) If $\mathbf{q}_i$ is an in-contact configuration, we compute the Euclidean distance from $\mathbf{o}$ to $\mathbf{q}_i$, and return it as the PD; i.e. $\mathrm{PD}(\mathfrak{A}, \mathfrak{B}) = \|\mathbf{o} - \mathbf{q}_i\|$. The algorithm terminates.
   (b) If $\mathbf{q}_i$ is a collision-free configuration, $\mathbf{q}_i$ becomes the source configuration ($\mathbf{q}^s \equiv \mathbf{q}_i$), and the origin becomes the target configuration ($\mathbf{q}^t \equiv \mathbf{o}$). Then we go to step 2.
   (c) If $\mathbf{q}_i$ is an in-collision configuration, we obtain a proper contact configuration by setting $\mathbf{q}_i$ to a target configuration ($\mathbf{q}^t \equiv \mathbf{q}_i$), and the previous contact configuration $\mathbf{q}_{i-1}$ becomes the source configuration ($\mathbf{q}^s \equiv \mathbf{q}_{i-1}$). Then we go to step 2.
5. Steps 2-4 are repeated until the algorithm terminates.

Note that this algorithm always terminates since it is a local optimization on the LCS, and a locally optimal solution is obtained whenever $\mathbf{q}_i$ is in-contact. Fig.8 illustrates this algorithm.

4 Out-Projection Using Continuous Collision Detection

Our algorithm iteratively updates a sample configuration on the contact-space to find a locally optimal configuration. This update process requires a method of projecting the current in-contact or collision-free configuration on to another configuration on the contact-space. This out-projection is achieved by a variant of continuous collision detection (CCD).

4.1 Continuous Collision Detection

Let $\mathfrak{A}$ and $\mathfrak{B}$ be two polygon-soup models in 3D, where $\mathfrak{A}$ is movable by a translation $\mathbf{M}(t)$ and $\mathfrak{B}$ is fixed. The source and target configurations of $\mathfrak{A}$ are $\mathbf{q}^s$ and $\mathbf{q}^t$ at times $t = 0$ and $t = 1$ respectively. We also define $\mathfrak{A}(t) \equiv \mathbf{M}(t)\mathfrak{A}$, and $\mathbf{q}(t)$ represents a configuration of $\mathfrak{A}(t)$. Then the continuous collision detection (CCD) problem can be formulated as a search for the first time of contact (ToC), τ, between $[0, 1]$, if it exists:

$$\tau = \min\{t \in [0, 1] \mid \mathfrak{A}(t) \cap \mathfrak{B} \neq \emptyset\}. \tag{5}$$

There are many methods for CCD, but our choice is conservative advancement (CA) [Lin93, Mir96, ZLK06, ZRLK07, TKM09] which is known to be fastest in practice. Like other CCD algorithms, CA takes the source $\mathbf{q}^s$ and target $\mathbf{q}^t$ configurations of an object $\mathfrak{A}$, and computes the first time of contact when $\mathfrak{A}$ linearly interpolates from $\mathbf{q}^s$ to $\mathbf{q}^t$ in the configuration space. For a convex polytope, CA computes a lower bound on τ by repeatedly advancing $\mathfrak{A}$ by Δt toward $\mathfrak{B}$ until collision occurs. The value of Δt is chosen to correspond to a lower bound on the closest distance $d(\mathfrak{A}(t), \mathfrak{B})$ between $\mathfrak{A}(t)$ and $\mathfrak{B}$, and an upper bound μ on the motion of $\mathfrak{A}(t)$ projected on to the direction of $d(\mathfrak{A}(t), \mathfrak{B})$ per second:

$$\Delta t \leq \frac{d(\mathfrak{A}(t), \mathfrak{B})}{\mu}. \tag{6}$$

The first time of contact τ is the sum of the time-steps Δt before collision; i.e. $\tau = \sum \Delta t$. Whereas the CA algorithm applies to convex polytopes, we need to use the modified version of the C^{2}A algorithm proposed by Tang et al. [TKM09] for polygon-soup models. The C^{2}A algorithm uses a bounding volume hierarchy (BVH) based on swept sphere volumes (SSVs) [LGLM00] to control the depth of the recursive BVH traversal during iterations of the algorithm, which reduces the computation time significantly. Since our PD problem is restricted to translational motion, we can simplify C^{2}A and make it faster. We present more details of this technique in Sec. 4.2.

When we have found τ, we can perform a static proximity query [LGLM00] to find all the contact features between $\mathfrak{A}(\tau)$ and $\mathfrak{B}$, such as VF and EE contacts. These will be used to construct the boundary of the local contact space in Sec. 6.

4.2 Translational Continuous Collision Detection

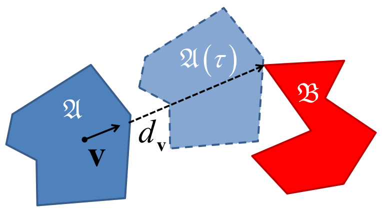


Figure 2: **Minimal directional distance along v between two objects.**

If $\mathfrak{A}$ is moving with a constant translational velocity $\mathbf{v}$, then we can call the shortest distance between $\mathfrak{A}$ and $\mathfrak{B}$ in the direction of $\mathbf{v}$ the minimal directional distance (MDD) $d_\mathbf{v}(\mathfrak{A}, \mathfrak{B})$, as illustrated in Fig. 2. The time at which $\mathfrak{A}$ contacts $\mathfrak{B}$ can be calculated as follows:

$$\tau = \frac{d_\mathbf{v}(\mathfrak{A}, \mathfrak{B})}{\|\mathbf{v}\|}. \tag{7}$$

If $\tau < 1$, then $\mathfrak{A}$ and $\mathfrak{B}$ will collide at τ; otherwise, they are collision-free during the entire time-step.

[CLR$^+$06] presented a method of computing the MDD between convex polytopes based on Minkowski sums and ray-shooting. However, no algorithm for computing the MDD between polygon-soup models has been reported. We propose a simple method in which we construct the BVHs of polygon-soup models and recursively compute the MDD between pairs of nodes pairs in the BVHs; this is similar to the computation of Euclidean distance based on BVHs. Thus computing $d_\mathbf{v}(\mathfrak{A}, \mathfrak{B})$ boils down to computing the MDD between pairs of nodes in the BVHs (i.e. these nodes are bounding volumes (BVs) or triangles). We will explain how to compute the MDD between two triangles $\triangle_{\mathfrak{A}}$ and $\triangle_{\mathfrak{B}}$; and it should be apparent that a similar method can be used between BVs.

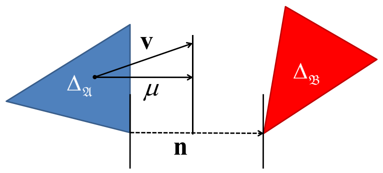


Figure 3: **Translational CA for triangles.**

The ToC τ' of two triangles under translational motion $\mathbf{v}$ can be found by repeated application of Eq. 6. The motion bound μ is simply $\mathbf{v} \cdot \mathbf{n}$, where $\mathbf{n}$ connects two closest points on the triangles, as illustrated in Fig. 3. Then the MDD between $\triangle_{\mathfrak{A}}$ and $\triangle_{\mathfrak{B}}$ is written $d_{\mathbf{v}}(\triangle_{\mathfrak{A}}, \triangle_{\mathfrak{B}}) = \tau'\|\mathbf{v}\|$. We recursively compute the MDD between BV pairs or triangles pairs in the BVHs, and use Eq. 7 to obtain the ToC between the polygon-soup models $\mathfrak{A}$ and $\mathfrak{B}$.

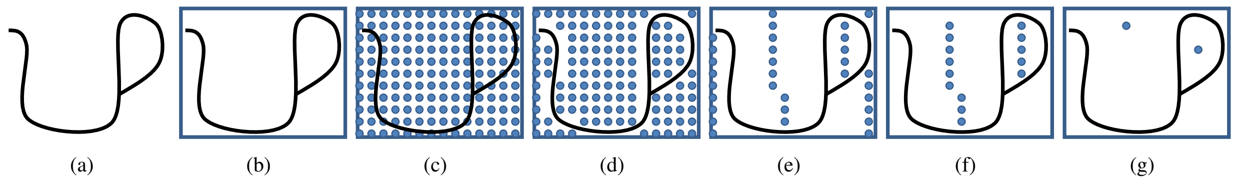


Figure 4: **Finding maximally clear configurations.** (a) The given model. (b) AABB of the model. (c) Voxelize the AABB and compute the distance to the model from each grid position. (d) Remove the grid points corresponding to small distances. (e) Compare neighboring grid points and remove those with smaller distances by scanning along the x- and y-axes. (f) Remove the grids on the boundary of AABB. (g) Compare neighboring grid points to remove those with smaller distances, and find maximally distant configurations in the x-, y- and z-axes.

5 Finding a Collision-Free Configuration

To obtain a non-trivial solution to Eq. 5, the source configuration $\mathbf{q}^s$ needs to be collision-free; otherwise, τ is trivially zero. In the context of our PD computation, $\mathbf{q}^s$ is unknown, whereas the target configuration $\mathbf{q}^t$, which is an in-collision configuration, is an input to the PD problem. We will introduce several methods of finding a collision-free source configuration $\mathbf{q}^s$. This is crucial to our PD computation, since our algorithm is a local optimization and starts from the contact configuration computed by CCD.

5.1 Centroid Difference

When no prior information is available about the interpenetration of the objects, the penetration direction can be estimated from the centroids of the objects. This direction can then be used to place the moving object in an interpenetration-free configuration, which can be expressed as follows:

$$\mathbf{q}^s = \mathbf{q}^{\mathfrak{A}} + r\frac{\mathbf{o}^{\mathfrak{A}} - \mathbf{o}^{\mathfrak{B}}}{||\mathbf{o}^{\mathfrak{A}} - \mathbf{o}^{\mathfrak{B}}||}$$

, where $\mathbf{q}^{\mathfrak{A}}$ is the initial configuration of $\mathfrak{A}$, $\mathbf{o}^{\mathfrak{A}}$ and $\mathbf{o}^{\mathfrak{B}}$ are the centroids of $\mathfrak{A}$ and $\mathfrak{B}$ respectively, and $2r$ is an upper bound on the size of the object. For instance, r might be the radius of a sphere known to enclose an object. Fig. 5 shows a collision-free configuration determined from the centroid difference.

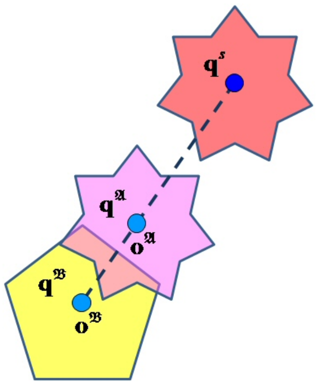

$\mathbf{q}^s$
$\mathbf{q}^{\mathfrak{A}}$
$\mathbf{o}^{\mathfrak{A}}$
$\mathbf{q}^{\mathfrak{B}}$
$\mathbf{o}^{\mathfrak{B}}$

Figure 5: **A collision-free configuration from the centroid difference.** Obstacle $\mathfrak{B}$ (yellow), initial in-collision configuration of $\mathfrak{A}$ (magenta), and the collision-free configuration of $\mathfrak{A}$ (red).

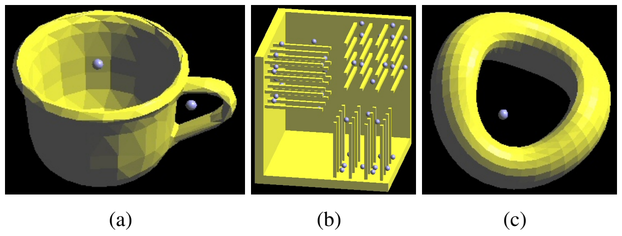


Figure 6: **Maximally clear configurations for the Cup, Grate, and Distorted-torus models.** The small gray spheres show the positions of the maximally clear configurations.

5.2 Maximally Clear Configuration

Although the centroid difference allows us to obtain a collision-free configuration, that configuration may be quite different from the optimal PD configuration for a complicated object with a lot of concavities or many holes. To obtain a collision-free configuration for objects of this sort, we find points in space that have maximal clearance. This preprocess allows objects to be positioned without interpenetration. These points correspond to the points on the boundary of external Voronoi regions of the object, which are equidistant from at least two points on the surface of the model. These maximally clear configurations are appropriate candidates for collision-free configurations since they correspond to locally maximum likelihoods of a collision-free state (see Fig. 6). A similar strategy has been used in workspace sampling in motion planners [Lat91]. Since the construction of a generalized Voronoi diagram for a polygonal model is quite hard [HCK$^+$99] we propose a simple algorithm based on a voxelization of space (also see Fig. 4):

1. Compute the axis-aligned bounding box (AABB) of the static object, and voxelize the AABB with a grid.
2. Calculate distance to the object from each point in the grid.
3. Eliminate configurations with small distance values, since these nearly correspond to collisions.
4. Compare neighboring configurations and eliminate those with shorter distances; and find maximally distant configurations in one and two dimensions.
5. Remove any configurations remaining on the boundary of the AABB.
6. Again compare neighboring configurations and eliminate those with shorter distances, thus identifying the maximally distant configurations in full dimensions.

Note that the above procedure only computes a subset of the points on the Voronoi boundary, which are those likely to have maximal clearance.

5.3 Motion Coherence

Application of our PD algorithm are likely to exhibit motion coherence: for instance, in rigid or articulated body dynamics, a series of collision-free or contact configurations are calculated as a function of time, we can exploit the underlying motion coherence to find a good initial source configuration. A simple way of doing this is to cache a sequence of collision-free configurations and choose the one closest to a given in-collision configuration.

5.4 Random Configuration

In a more general setting, in which we do not have any knowledge of $\mathbf{q}^s$, we can randomly sample $\mathbf{q}^s$ in the free configuration space [ZKM07] or simply locate $\mathbf{q}^s$ at infinity.

5.5 Sampling-based Search

The methods of finding a suitable collision-free configuration from which to begin out-projection that we have just described may still generate a configuration far from the optimal. A sampling-based search algorithm can be employed to refine an initial collision-free configuration. By performing collision detection on a series of configurations sampled on a line from the given collision configuration $\mathbf{o}$ to $\mathbf{q}_0^f$, as illustrated in Fig.7. This sampling process terminates whenever a collision-free configuration is found.

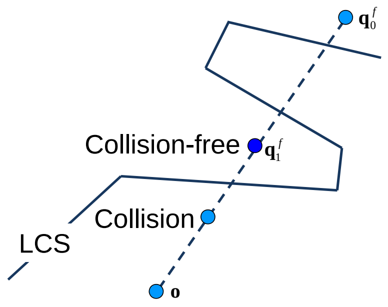


Figure 7: **Sampling-based search for a collision-free configuration.** The input collision configuration is $\mathbf{o}$ and $\mathbf{q}_0^f$ is an initial collision-free configuration. The first sample on the line from $\mathbf{o}$ to $\mathbf{q}_0^f$ is determined to be in-collision, but the second sample $\mathbf{q}_1^f$ is collision-free.

6 Iterative Optimization

Any sampled configuration $\mathbf{q}$ in the contact space can be used as an estimate of the PD by computing its distance from the origin; i.e. $\mathrm{PD}(\mathfrak{A}, \mathfrak{B}) = \|\mathbf{o} - \mathbf{q}\|$. When we have to deal with a highly non-convex object, finding a good candidate for $\mathbf{q}$ taxes the strategies suggested in Sec. 5. Thus we need to refine the sample to get a PD closer to the optimum, which we achieve by a *walk* in contact space. We use the result found by one of the techniques described in Sec. 5 as the start of this walk, and then build a local approximation to the contact space, and refine the configuration by in-projection. We repeat this process until a locally optimal solution is obtained.

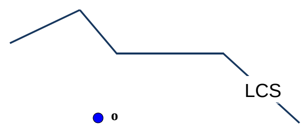
6.1 Contact-Space Localization and In-Projection

The out-projection process explained in Sec. 4 generates an in-contact configuration, which corresponds to a pair of contact features, one in each of $\mathfrak{A}$ and $\mathfrak{B}$. In general, this feature pair forms a vertex/face (VF), a face/vertex (FV) or an edge/edge (EE) primitive contact, from which we can construct a local contact space (LCS). Then, for the ith primitive contact, we determine the equation of a contact plane, $\mathbf{j}_i\mathbf{q} = c_i$ where $\mathbf{j}_i$ and c_i respectively are the row vector of coefficients and the scalar bias of the ith plane equation.

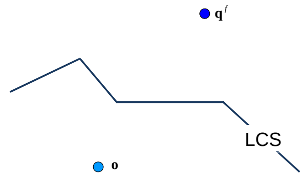
In some cases, this contact plane degenerates to a lower-dimensional subset of configuration space, such as a line or a point. For example, contact between two collinear edges or a VE contact produces a contact line equation, and a VV contact produces a point. Our algorithm simply ignores VV contacts and collinear edges, since projection on to a space of reduced dimensionality is unlikely to improve the PD. In the case of a VE contact, we generate several VF primitive contacts for all the faces that share the edge in the VE, and intersect them. The result is likely to be over-constrained since these contacts should be combined using the union operator [ZKM07]. However, VE occurs rarely and the use of intersection simplifies implementation and improves performance. All other non-primitive contacts are also converted to several primitive contacts.

By stacking all the contact equations into a matrix, we can construct a system of linear equations:

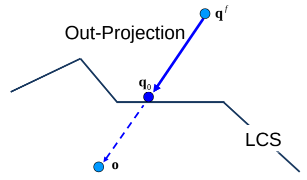
$$\mathbf{J}\mathbf{q} = \mathbf{c}. \tag{8}$$



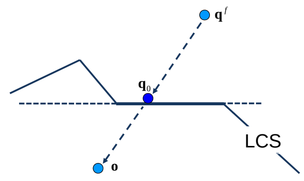
(a) Local contact space and the origin



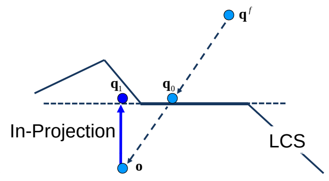
(b) Collision-free configuration $\mathbf{q}^f$



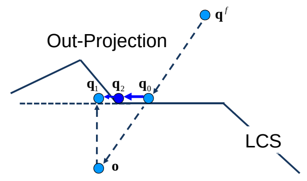
(c) Out-projection from $\mathbf{q}^f$ to $\mathbf{o}$ to obtain $\mathbf{q}_0$



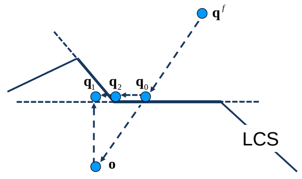
(d) Construction of an LCS around $\mathbf{q}_0$



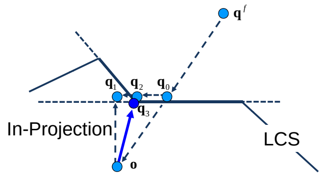
(e) In-projection on to the LCS to obtain $\mathbf{q}_1$



(f) Out-projection from $\mathbf{q}_0$ to $\mathbf{q}_1$ to obtain $\mathbf{q}_2$



(g) LCS around $\mathbf{q}_2$



(h) In-projection on to the LCS to obtain $\mathbf{q}_3$

Figure 8: **Iterative optimization of a PD.**

We can then formulate in-projection as a minimization of the squared Euclidean distance between the origin $\mathbf{o}$ and a configuration $\mathbf{q}$ on the LCS, under the contact constraints:

$$\text{Minimize } \|\mathbf{q}\|^2 \text{ subject to } \mathbf{J}\mathbf{q} \geq \mathbf{c}, \tag{9}$$

where $\mathbf{J}\mathbf{q} \geq \mathbf{c}$ constrains $\mathbf{q}$ to lie either on the boundary of the LCS or outside it since the system of equations $\mathbf{J}\mathbf{q} = \mathbf{c}$ represents the LCS.

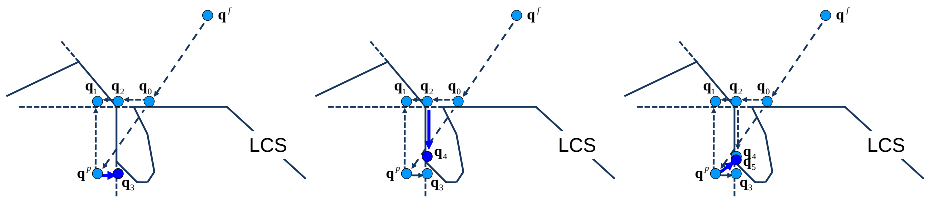
It is known [RKC02] that Eq.9 is equivalent to a linear complementarity problem (LCP), formulated as a search for a value of λ which satisfies the following complementarity condition:

$$\begin{cases} -\frac{1}{4}\mathbf{J}\mathbf{J}^T\lambda + \mathbf{c} \geq \mathbf{0} \\ \lambda \geq \mathbf{0} \\ (-\frac{1}{4}\mathbf{J}\mathbf{J}^T\lambda + \mathbf{c})^T\lambda = 0 \end{cases} \tag{10}$$

Then, $\mathbf{q} = \frac{1}{4}\mathbf{J}^T\lambda$.

There are several methods of solving an LCP such as Lemke's and Dantzig's algorithm [CPS09]; however we choose a projected Gauss-Seidel method for simplicity and stability [JAJ98]. Then the LCP reduces to the search for a solution of the following linear system:

$$\mathbf{J}\mathbf{J}^T\lambda = 4\mathbf{c}. \tag{11}$$



(a) Collision at $\mathbf{q}_3$ (b) Out-projection to obtain $\mathbf{q}_4$ (c) In-projection to obtain $\mathbf{q}_5$

Figure 9: **Iterative optimization of a PD for a more complicated LCS.**

And the Gauss-Seidel iteration is:

$$\lambda_{k+1} = (\mathbf{D}+\mathbf{L})^{-1}(4\mathbf{c} - \mathbf{U}\lambda_k), \tag{12}$$

where $\mathbf{D}+\mathbf{L}+\mathbf{U} = \mathbf{J}\mathbf{J}^T$, and the matrices $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ respectively represent the diagonal, strictly lower triangular, and strictly upper triangular parts of the coefficient matrix $\mathbf{J}\mathbf{J}^T$, and k denotes the iteration number. When a component of λ becomes negative during the iteration, we set it to zero: this is the *projection* step in the projected Gauss-Seidel (PGS) algorithm.

A Gauss-Seidel iteration with an arbitrary symmetric positive definite matrix [GVL96] is known to converge. $\mathbf{J}\mathbf{J}^T$ is positive semidefinite since it is symmetric and $\mathbf{x}^T\mathbf{J}\mathbf{J}^T\mathbf{x} = (\mathbf{J}^T\mathbf{x})^T\mathbf{J}^T\mathbf{x} = ||\mathbf{J}^T\mathbf{x}||^2 \geq 0$ for $\mathbf{x} \in \mathbb{R}^n$. We remove the redundant rows of $\mathbf{J}$ to make $\mathbf{J}$ full rank, and thereby promote $\mathbf{J}\mathbf{J}^T$ from semidefinite to positive definite since now $\mathbf{x}^T\mathbf{J}\mathbf{J}^T\mathbf{x} = (\mathbf{J}^T\mathbf{x})^T\mathbf{J}^T\mathbf{x} = ||\mathbf{J}^T\mathbf{x}||^2 > 0$ for all non-zero vectors $\mathbf{x} \in \mathbb{R}^n$. Therefore, the Gauss-Seidel component of our algorithm always converges.

We store the contact features as a list, sorted by distance from $\mathbf{o}$ to the corresponding contact plane. If a model is complicated, the number of contact features can become excessively high, and we select between only 10 and 30 contact feature pairs to reduce the complexity of solving Eq. 10. This produces satisfactory results in practice.

6.2 Iteration

As described in the overview presented in Sec. 3.2, we iteratively optimize a sample to refine the penetration depth. Now we describe more details of this process, which relates to Fig.8:

(a) An in-collision configuration (or the origin $\mathbf{o}$) is given as input (Fig. 8(a)).

(b) Select a collision-free configuration $\mathbf{q}^f$ using the technique described in Sec 5 (Fig. 8(b)).

(c) Using the configurations $\mathbf{q}^f$ as source and $\mathbf{o}$ as target, perform out-projection to obtain a contact configuration $\mathbf{q}_0$ (Fig. 8(c)).

(d) Construct an LCS around the contact configuration $\mathbf{q}_0$ (Fig. 8(d)). In this example, the LCS consists of only a single contact plane.

(e) Perform in-projection on to the LCS to obtain $\mathbf{q}_1$ (Fig. 8(e)). A proximity query is used to classify the current configuration $\mathbf{q}_1$ as contact, separation, or penetration. In this example, $\mathbf{q}_1$ corresponds to penetration; so the next step is to find a valid in-contact configuration.

(f) Again perform out-projection to obtain $\mathbf{q}_2$ (Fig. 8(f)). This requires a non-colliding source configuration as well as an in-collision target configuration. The source configuration (i.e. $\mathbf{q}_0$) was obtained from the previous out-projection, and the current configuration $\mathbf{q}_1$ can be used as the target configuration.

(g) Construct the LCS around $\mathbf{q}_2$ (Fig. 8(g)). This is a convex cone (the intersection of two contact planes), since $\mathbf{q}_2$ consists of two contact feature pairs.

(h) In-projection is performed again (Fig. 8(h)). The new configuration $\mathbf{q}_3$ is in-contact and the PD that will be returned is $||\mathbf{q}_3 - \mathbf{o}||$.

Fig. 9 shows a more complicated scenario. The first two steps obtain $\mathbf{q}_1$ and $\mathbf{q}_2$, as before. But this time the proximity query at $\mathbf{q}_3$ detects penetration (Fig. 9(a)). Thus we perform out-projection to obtain $\mathbf{q}_4$ (Fig. 9(b)), construct the LCS, and run in-projection again to get the final configuration $\mathbf{q}_5$ (Fig. 9(c)).

6.3 Estimation of Local Penetration Depth

When two objects intersected in multiple disjoint regions, applications may require separate information about all the penetrations, i.e. local PDs instead of a single global PD. For instance, in rigid-body dynamics, a local PD, defined as the locally deepest penetrating points of two colliding objects [GBF03, TLK09], is required to compute torques or to stabilize the simulation. [GBF03] compute local PDs from distance fields, whereas [TLK09] use two-sided Hausdorff distance. We propose an alternative method to derive local PDs from the global PD information that our algorithm computes. Our method of local PD estimation is based on the hypothesis that a pair of locally deepest penetrating points, one from each object, coincide when they are translated by the amount of the global PD. Moreover, the direction of local PD corresponds to the contact normal of the points.

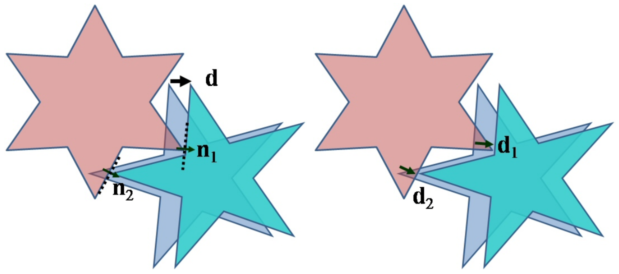
More specifically, we compute the local PD $\mathbf{d}_i$ of the ith interpenetrating region as follows (also see Fig. 10):

1. Given two overlapping models $\mathfrak{A}$ and $\mathfrak{B}$, compute their global PD $d \equiv \mathrm{PD}(\mathfrak{A}, \mathfrak{B})$ using the PD algorithm already presented.
2. Translate $\mathfrak{A}$ by $\mathbf{d}$ to become $\mathfrak{A}(\mathbf{d})$, and compute the contact normal $\mathbf{n}_i$ for each contact i between $\mathfrak{A}(\mathbf{d})$ and $\mathfrak{B}$.
3. Project $\mathbf{d}$ on to each $\mathbf{n}_i$ to get the local PD $\mathbf{d}_i$; i.e. $\mathbf{d}_i = (\mathbf{d} \cdot \mathbf{n}_i)\mathbf{n}_i$.

Note that our definition of local PD is not equivalent to previous definition of [GBF03, TLK09], but it is computationally efficient. Our local PD algorithm is a simple by-product of our global PD algorithm, and does not require any costly and memory-intensive precomputation of distance fields unlike [GBF03], and no expensive Boolean intersection is required unlike [TLK09].

7 Results and Discussion

We now present the results obtained by our PD computation algorithm in various scenarios, and discuss some of the implementation issues.



(a) Global PD, $\mathbf{d}$, and two contact normals $\mathbf{n}_1$, $\mathbf{n}_2$ (b) Two local PDs, $\mathbf{d}_1$ and $\mathbf{d}_2$

Figure 10: **Estimation of local PDs from the global PD.** The red ($\mathfrak{B}$) and blue ($\mathfrak{A}$) objects are placed in a penetrating configuration. The blue object is translated to the cyan one by the global PD $\mathbf{d}$, and the two contact normals $\mathbf{n}_1$ and $\mathbf{n}_2$ are obtained at the contact. Projecting $\mathbf{d}$ on to $\mathbf{n}_1$ and $\mathbf{n}_2$ gives the two local PDs $\mathbf{d}_1$ and $\mathbf{d}_2$.

7.1 Implementation and Benchmarks

We implemented our PD computation algorithm using Microsoft Visual C++ 8.0 under the Windows XP operating system. We tested the algorithm on a PC equipped with an AMD 2.22GHz CPU and 2Gb of RAM. The benchmarking models that we used in these experiments are shown in Fig. 11. The complexities of these models range from 0.54k to 174k triangles, and their topologies contain many holes and self-intersections (i.e. polygon-soups). We constructed a number of benchmarking scenarios for these models, including random configurations, predefined sequences, and collisions within a rigid-body dynamics simulation.

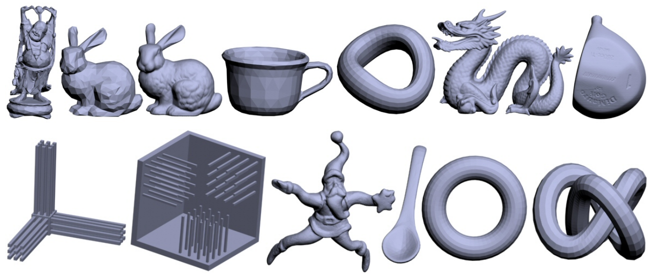


Figure 11: **Benchmarking models with triangle counts. Top row:** Buddha (10k), Bunny1 (1k), Bunny2 (40k), Cup (1k), Distorted-torus (1.33k), Dragon (174k), Golf-club (105k) **Bottom row:** Grate1 (0.54k), Grate2 (0.94k), Santa (152k), Spoon (1.34k), Torus (2k), and Torusknot (3K).

7.1.1 Random Configuration Scenario

We created 100 random configurations for several of the models, including the Torusknot, Buddha, Bunny2 and Dragon. We place a model at a fixed configuration, and translated a copy of the same model through a distance equal to half the size of its bounding volume, while changing its orientation, all at random creating many deeply penetrating configurations. Fig. 12 shows an example in which two copies of the Torusknot model intersect.

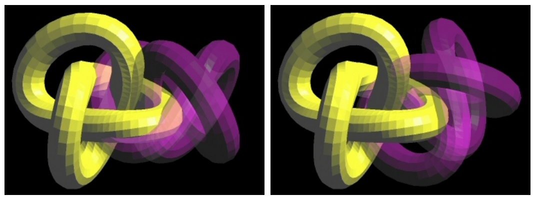


Figure 12: **Two Torusknot models in a random configuration.**

We use a centroid difference to find the initial collision-free configuration. The performance of our algorithm is characterized in Table 1. These values are averaged for all the collision frames; the number of contacts means the number of contact feature pairs used to construct the LCS and the number of iterations is the total number of in- and out- projections for computing the PD. In Fig. 14, we also show the number of contact feature pairs, number of iterations, and computation time for two intersecting Torusknots in random configurations.

Table 1: **Performance of our algorithm with random configurations.**

Model	Time (msec)	No. of Contacts	No. of Iterations
Torusknot	5.53	4.84	2.81
Buddha	6.23	5.04	1.92
Bunny2	7.55	8.25	2.16
Dragon	12.76	11.92	2.29

For the more topologically challenging models such as Grate1, Grate2 and Distorted-torus, we use maximally clear configurations and sampling-based search scheme to find the initial collision-free configurations, because of their very concave geometries. Results are shown in Fig. 13.

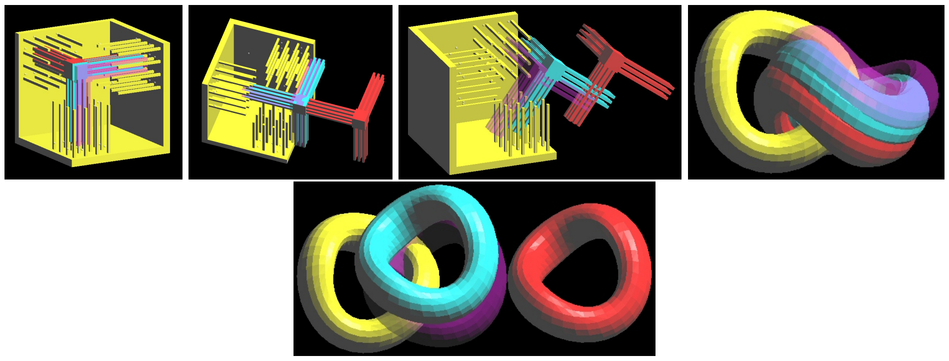
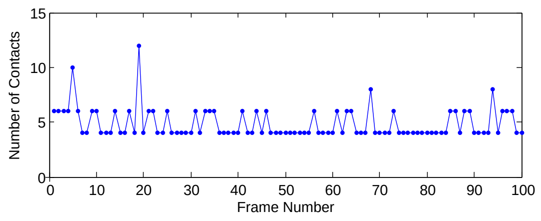
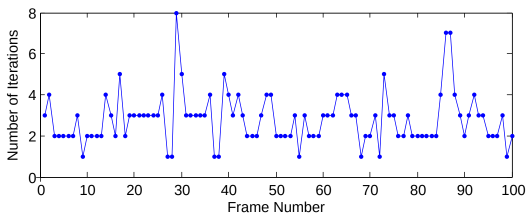


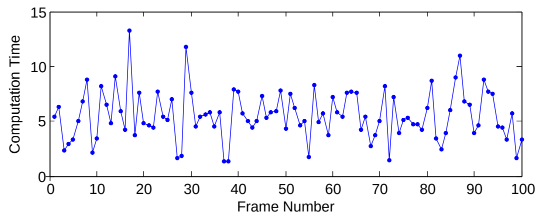
Figure 13: **Using maximally clear configurations and sampling-based search to determine collision-free configurations for the Grate and Distorted-torus models.** Obstacle $\mathfrak{B}$ is yellow, and the input in-collision configuration of $\mathfrak{A}$ is semitransparent magenta. The initial collision-free configuration of $\mathfrak{A}$ is red, and the configuration of $\mathfrak{A}$ translated by PD is shown in cyan.



(a) Number of contact features



(b) Number of iterations



(c) Computation time (msec)

Figure 14: **Accuracy of results for two Torusknot models in 100 random configurations.**

In order to assess the accuracy of our PD algorithm, we used Lien's open source library [Lie08b] to compute near-exact Minkowski sums and PDs for the same models. Lien's method is near-exact in the sense that it ignores low-dimensional boundaries of the Minkowski sums which have a volume close to zero. Fig. 15 shows the PDs and the computational times for two Bunny1 models and two Distorted-torus models using our method and Lien's.

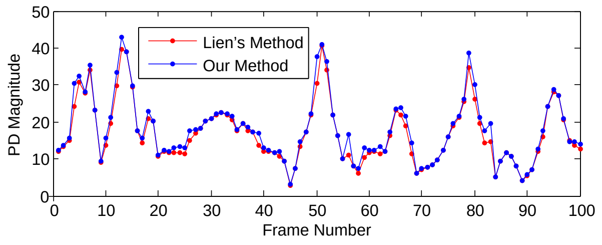
The results are very close, even though our method is about 2000 times faster than Lien's. A quantitative comparison can be obtained by defining a metric for the relative error between approximate and exact PDs as follows:

$$\mathbf{e}_{PD,relative1} = \frac{\|\mathbf{PD}_{appoximate} - \mathbf{PD}_{exact}\|}{\max w(\mathfrak{A} \oplus -\mathfrak{B})}$$

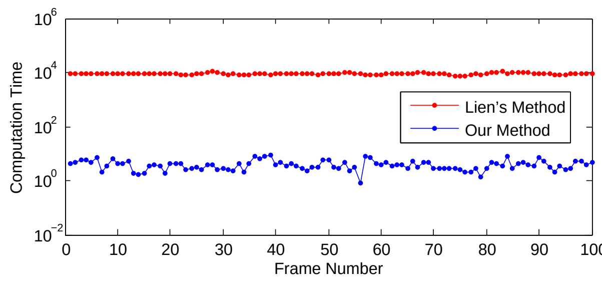
, where w is the width of an object in a given direction $\mathbf{n}$, defined as the distance between supporting planes normal to $\mathbf{n}$. Since the PD is the minimum distance from the origin to the boundary of the Minkowski sums, $\max w(\mathfrak{A} \oplus -\mathfrak{B})$ is an upper bound on the PD. However, since this metric depends on orientation, it can widely vary. An alternative metric which is unaffected by orientation is:

$$\mathbf{e}_{PD,relative2} = \frac{\|\mathbf{PD}_{approximate} - \mathbf{PD}_{exact}\|}{2\overline{v}(\mathfrak{A}) + 2\overline{v}(\mathfrak{B})}$$

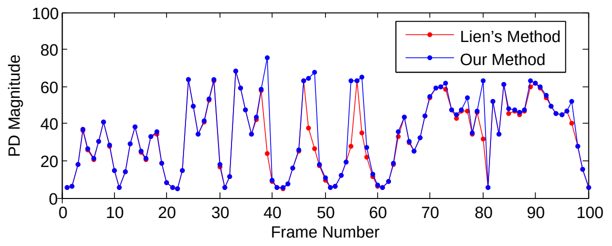
, where $\overline{v}$ denotes the average magnitude of the vertices of an object, i.e. $\overline{v} = \frac{1}{N}\sum_i \|\mathbf{v}_i\|$. If the objects are spheres, $\mathbf{e}_{PD,relative1} = \mathbf{e}_{PD,relative2}$, since the $\overline{v}$s would be the radii of the spheres. If the PDs from Lien's method are considered to be exact, then the errors in the results from our algorithm are listed in Table 2.



(a) PD Magnitude (Bunny1)



(b) Computational Time (Bunny1)



(c) PD Magnitude (Distorted Torus)

Figure 15: **Graphs of PD comparisons using our method and Lien's method [Lie08b] at random configurations.**

Table 2: **Average PD errors**

Model	Mean error (%)	Median error (%)
Bunny1	0.500	0.165
Distorted-Torus	1.165	0.066

7.1.2 Predefined Path Scenario

Using the Havok$^{\mathrm{TM}1}$ a rigid-body dynamics simulator, we generated paths for the Dragon (Fig. 16), and Spoon and Cup models (Fig. 17), and tested our PD algorithm along these paths. Fig. 18 shows the number of contact feature pairs, the number of iterations, and the computational time for the Dragon model. One Dragon is fixed in space and a copy falls under gravity, as shown in Fig. 16. The centroid difference explained in Sec.5.1 is used to find an initial collision-free configuration.

Fig. 20 shows scenes from the motion of the Spoon and Cup models along a predefined path. Collision-free configurations were found using the centroid difference (a∼d) and using motion coherence (e∼j). Fig. 19 shows the results when maximally clear configurations were used for the same scene. Fig. 6(a) shows the maximally clear configuration for the Cup. The centroid difference can generate a poor collision-free configuration when the underling geometry and topology are complicated, as demonstrated in Fig. 20(b). But in the same situations, the maximally clear configurations are very satisfactory, as demonstrated in Fig. 20(e)∼(j). The performance of our algorithm for predefined paths is summarized in Table 3.

[1]`http://www.havok.com`

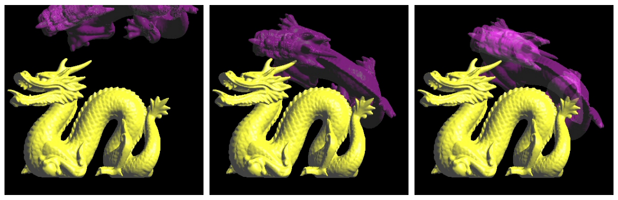


Figure 16: **Path for the Dragon model generated by a physics simulator.**

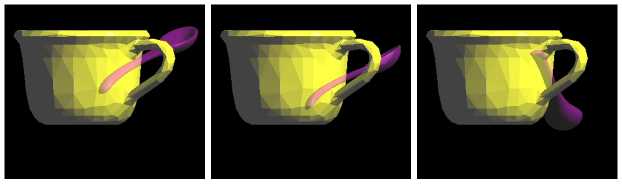


Figure 17: **Path of the Spoon model relative to Cup generated by a physics simulator.**

7.1.3 Dynamics Scenario

We integrated our PD algorithm into rigid-body dynamics simulations involving the Torus, Bunny2, Golf-club and Santa models (see Fig. 21). We used impulse-based dynamics [GBF03] to simulate rigid-body dynamics based on the local PD method presented in Sec. 6.3. The local PDs are used to stablize the simulation and to resolve contacts and collisions. In this scenario, most collisions do not create deep penetrations since collisions are resolved immediately. As a result, the average PD computation is shorter, for example, than that required for a random configuration. We use maximally clear configurations, motion coherence and centroid differences to find initial collision-free configurations. A performance summary is given in Table 4. Note that an additional time is required for local PD computations.

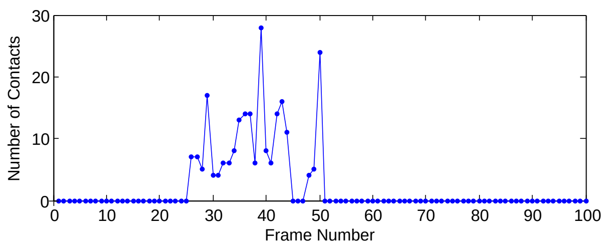
7.2 Discussion

There are some limitations to our algorithm. Our method only approximates an upper bound, which can depend on the initial collision-free configuration, as Fig. 20(b) shows. Thus the quality of the initial collision-free configuration is critical. We have presented several methods of generating an initial collision-free configuration, and now summarize our experiences:

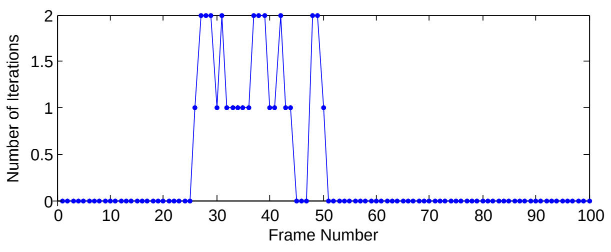
- Maximally clear configurations are suitable for strongly non-convex objects of high genus, such as the Grate, Torus and Cup in Figs. 13 and 19, since a collision-free configuration can be located inside a concavity.

Table 3: **PD performance on predefined paths**

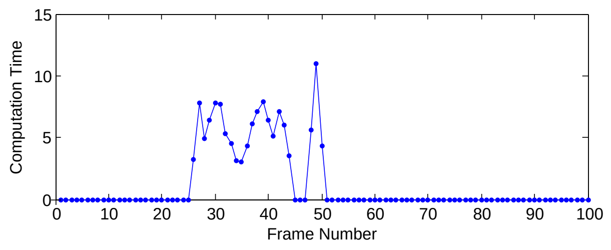
Models	Time (msec)	No. of Contacts	No. of Iterations
Spoon and Cup	0.96	4.49	1.02
Buddha	3.50	5.07	1.29
Dragon	5.84	10.32	1.45



(a) Number of Contact Features

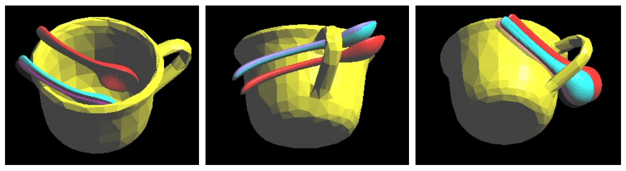


(b) Number of Iterations



(c) Computation Time (msec)

Figure 18: **Graphs of PD computation using two colliding Dragon models moving on predefined paths.**



(a) (b) (c)

Figure 19: **PD results using maximally clear configurations for Spoon and Cup models moving on predefined paths.**

- Sampling-based search is useful for intricate or interlocking objects such as those shown in Fig. 13, but it is time-consuming.
- In a dynamics simulation, such as that shown in Fig. 21, motion coherence can be exploited to provide a good candidate for an initial configuration by using the configuration in the frame before a collision occurs. This can be achieved by caching.
- The centroid difference is good for near-convex objects, unless they are deeply penetrating. In that case, we need to try several directions of back-off.
- If the preprocessing cost of finding a maximally clear configurations is unaffordable, and no application-dependent information is available, random sampling can be employed.

None of these strategies guarantees a bound on the PD, since our algorithm uses local optimization; it may terminate far from the global optimum. However, the results of Sec. 7.1.1 show that these strategies work very well in practice, even for very challenging cases.

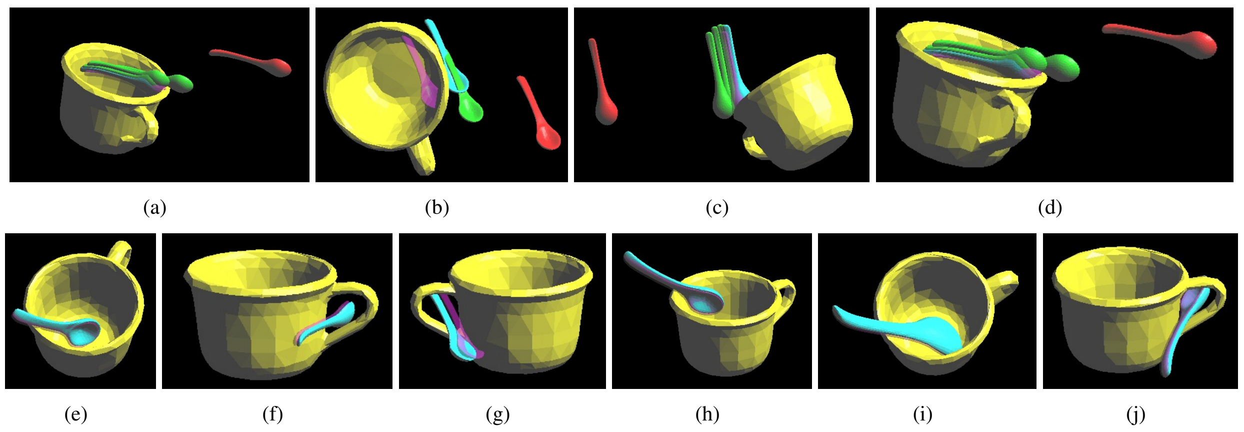


Figure 20: **PD results for the Spoon and Cup models moving on predefined paths.** Obstacles are shown in yellow, input in-collision configuration in semitransparent magenta, initial collision-free configurations in red, contact configurations from in- and out-projections in green, and the configuration translated by the PD is shown in cyan. In (a) $\sim$ (d), the centroid difference was used to find a collision-free configurations. In (e)$\sim$ (j), motion coherence was used.

8 Conclusions and Future Work

We have presented an interactive algorithm for computing the penetration depth of complicated polygon-soup models. Our method approximates the local contact space and iteratively performs in- and out-projections based on a Gauss-Seidel LCP solver and translational continuous collision detection. We have proposed several schemes for selecting an initial collision-free configuration which utilizes motion coherence, centroid difference, and maximally clear configurations. We showed the effectiveness of our PD algorithms in various scenarios. We also presented a method of computing a local PD, and integrated it with a dynamics simulation.

Future Work: We are interested in extending our algorithm to n-body PD problems, in which we need to separate multiple colliding bodies simultaneously. We would also like to extend our interactive algorithm to address the gener-

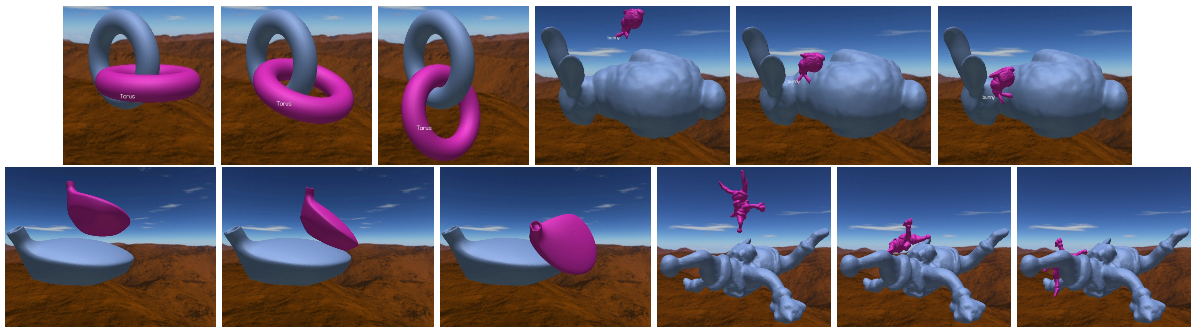


Figure 21: **Dynamic simulations results of pairs of Torus, Bunny2, Golf-club and Santa models using impulse-based dynamics [GBF03].**

Table 4: **Performance in the dynamics scenario.**

Models	Global PD (msec)	Local PD (msec)	Total (msec)
Torus	3.13	1.04	4.17
Bunny2	5.43	1.78	7.21
Golf-club	4.87	1.67	6.54
Santa	9.14	2.90	12.04

alized PD problem [ZKVM07, ZKM07, NPR07] in which both translational and rotational motions are possible. The key to this problem making amenable to our current framework is to find a way of linearizing the curved contact space. In addition, we are considering how our method might benefit other applications, such as haptic rendering and robot motion planning.

Acknowledgements

This research work was supported by the IT R&D program of MKE/MCST/KOCCA (2008-F-033-02, Development of Real-time Physics Simulation Engine for e-Entertainment).

References

[AGHp+00] Pankaj K. Agarwal, Leonidas J. Guibas, Sariel Har-peled, Alexander Rabinovitch, and Micha Sharir. Penetration depth of two convex polytopes in 3D. *Nordic J. Computing*, 7:227–240, 2000.

[Ben66] R. V. Benson. *Euclidean Geometry and Convexity*. McGraw-Hill, New York, NY, 1966.

[Ber01] G. Bergen. Proximity queries and penetration depth computation on 3d game objects. *Game Developers Conference*, 2001.

[Cam97] S. Cameron. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[CC86] S. A. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. In *Proc. of IEEE Inter. Conf. on Robotics and Automation*, pages 591–596, 1986.

[CLR+06] Yi-King Choi, Xueqing Li, Fengguang Rong, Wenping Wang, and Stephen Cameron. Computing the minimum directional distance between two convex polyhedra. *HKU CS Tech Report TR-2006-01*, 2006.

[CPS09] R.W. Cottle, J.S. Pang, and R.E. Stone. *The linear complementarity problem*. Society for Industrial & Applied Mathmatics, 2009.

[DHKS93] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

[FL01] S. Fisher and M. C. Lin. Fast penetration depth estimation for elastic bodies using deformed distance fields. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 2001*, 2001.

[GBF03] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. volume 22, pages 871–878, New York, NY, USA, 2003. ACM.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[Hac09] Peter Hachenberger. Exact minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica*, 55(2):329–345, 2009.

[HCK$^+$99] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of ACM SIGGRAPH 1999*, pages 277–286, 1999.

[HZLM02] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Technical Report TR02-004, Department of Computer Science, University of North Carolina, 2002.

[JAJ98] F. Jourdan, P. Alart, and M. Jean. A Gauss-Seidel like algorithm to solve frictional contact problems. *Computer Methods in Applied Mechanics and Engineering*, 155(1-2):31–47, 1998.

[KLM04] Young J. Kim, M. Lin, and D. Manocha. Incremental penetration depth estimation between convex polytopes using dual-space expansion. *IEEE Trans. on Visualization and Computer Graphics*, 10(1):152–164, 2004.

[KOLM02a] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation for physically-based animation. *Proc. of ACM Symposium on Computer Animation*, 2002.

[KOLM02b] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Fast penetration depth computation using rasterization hardware and hierarchical refinement. *Proc. of Workshop on Algorithmic Foundations of Robotics*, 2002.

[KOLM03] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha. Six-degree-of-freedom haptic rendering using incremental and localized computations. *Presence*, 12(3):277–295, 2003.

[Lat91] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[LGLM00] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation*, 2000.

[Lie08a] Jyh-Ming Lien. Covering minkowski sum boundary using points with applications. *Computer Aided Geometric Design*, 25(8):652–666, November 2008.

[Lie08b] Jyh-Ming Lien. A simple method for computing minkowski sum boundary in 3D using collision detection. In *The Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Guanajuato, Mexico, Dec 2008.

[Lin93] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, CA, December 1993.

[LM03] M. Lin and D. Manocha. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*, 2003.

[Mir96] Brian Vincent Mirtich. *Impulse-based dynamic simulation of rigid body systems*. PhD thesis, 1996.

[NPR07] G. Nawratil, H. Pottmann, and B. Ravani. Generalized Penetration Depth Computation based on Kinematical Geometry. *Geometry Preprint Series, Vienna Univ. of Technology, Tech. Rep*, 172, 2007.

[OG96] 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.

[Ong93] C.J. Ong. *Penetration distances and their applications to path planning*. PhD thesis, Michigan Univ., Ann Arbor., 1993.

[RKC02] S. Redon, A. Kheddar, and S. Coquillart. Gauss' least constraints principle and rigid body simulations. In *In proceedings of IEEE International Conference on Robotics and Automation*, pages 11–15, 2002.

[RL06] Stephane Redon and Ming C. Lin. A fast method for local penetration depth computation. *journal of graphics tools*, 11(2):37–50, 2006.

[SGG$^+$06] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha. Fast proximity computation among deformable models using discrete Voronoi diagrams. In *ACM SIGGRAPH*, pages 1144–1153. ACM New York, NY, USA, 2006.

[TKM09] M. Tang, Y. J. Kim, and D. Manocha. C^{2}A: controlled conservative advancement for continuous collision detection of polygonal models. In *IEEE International Conference on Robotics and Automation*, 2009.

[TLK09] Min Tang, Minkyoung Lee, and Young J. Kim. Interactive hausdorff distance computation for general polygonal models. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–9, New York, NY, USA, 2009. ACM.

[WZ09] R. Weller and G. Zachmann. Inner sphere trees for proximity and penetration queries. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[ZKM07] L. Zhang, Y. Kim, and D. Manocha. A fast and practical algorithm for generalized penetration depth computation. In *Proceedings of Robotics: Science and Systems*, 2007.

[ZKM08] L. Zhang, Y.J. Kim, and D. Manocha. Efficient cell labelling and path non-existence computation using c-obstacle query. *The International Journal of Robotics Research*, 27(11-12):1246–1257, Nov-Dec 2008.

[ZKVM07] L. Zhang, Y.J. Kim, G. Varadhan, and D. Manocha. Generalized penetration depth computation. *Computer-Aided Design*, 39(8):625–638, 2007.

[ZLK06] Xinyu Zhang, Minkyoung Lee, and Young J. Kim. Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer*, pages 749–760, 2006.

[ZM08] L. Zhang and D. Manocha. An efficient retraction-based RRT planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3743–3750, 2008.

[ZRLK07] Xinyu Zhang, Stephane Redon, Minkyoung Lee, and Young J. Kim. Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):15, 2007.