PolyDepth: Real-Time Penetration Depth Computation Using Iterative Contact-Space Projection

Article *in* ACM Transactions on Graphics · January 2012

DOI: 10.1145/2077341.2077346

5 authors, including:

Changsoo Je
Sogang University
19 PUBLICATIONS **325** CITATIONS
SEE PROFILE

Youngeun Lee
Ewha Womans University
7 PUBLICATIONS **132** CITATIONS
SEE PROFILE

Young J. Kim
Ewha Womans University
117 PUBLICATIONS **3,237** CITATIONS
SEE PROFILE

C. Je et al. PolyDepth: Real-time Penetration Depth Computation using Iterative Contact-Space Projection.
ACM Transactions on Graphics, Volume 33, Issue 1, Article 5, pp. 1-14, January 1, 2012. Also Presented in SIGGRAPH 2012.
DOI = 10.1145/2077341.2077346 http://doi.acm.org/10.1145/2077341.2077346

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

Changsoo Je
Ewha Womans University and Sogang University
and
Min Tang, Youngeun Lee, Minkyoung Lee and Young J. Kim
Ewha Womans University

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 using several methods such as centroid difference, maximally clear configuration, motion coherence, random configuration, and sampling-based search. 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.

Categories and Subject Descriptors: I.2.9 [**Artificial Intelligence**]: Robotics—*kinematics and dynamics*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*physically-based modeling*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*animation*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*animation*

General Terms: Animation, Dynamics

Additional Key Words and Phrases: Penetration Depth, Collision Detection, Polygon-Soups

Email: vision@sogang.ac.kr (C. Je)
{tangmin,kimy}@ewha.ac.kr (M. Tang, Y. J. Kim)
{youngeunlee,minkyounglee}@ewhain.net (Y. Lee, M. Lee).
Authors' addresses: C. Je, Ewha Womans University, Seoul, Korea and Sogang University, Seoul, Korea; M. Tang, Y. Lee, M. Lee, and Y. J. Kim (corresponding author), Ewha Womans University, Seoul, Korea.

DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

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 [Lin and Manocha 2003]. 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 [Cameron and Culley 1986], generalized penetration depth [Zhang et al. 2007b], pointwise penetration depth [Tang et al. 2009], growth distance [Ong 1993], and penetration volume [Weller and Zachmann 2009].

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 [Dobkin et al. 1993]. Many applications can benefit from PD computations. In physically-based animation, the PD can be used to locate the point of application for impulses [Guendelman et al. 2003], or used to find the time of contact in time-stepping methods [Kim et al. 2002]. In constraint-based dynamics, when penetration is unavoidable due to numerical errors, PD can be used to roll back from an invalid state (penetration) to a valid one (non-penetration) [Redon 2004]. Moreover, PD can be used for post-stabilization such as the Baumgarte stabilization in rigid-body dynamics to enforce non-penetration constraints. In virtual prototyping, the PD can be used to verify tolerances [Kim et al. 2004a]. The length of a PD can be used to determine the appropriate force feedback in six-degree-of-freedom haptic systems [Kim et al. 2003]. Retraction-based motion planning algorithms can use PD to find a collision-free sample in the narrow passage amongst obstacles [Zhang and Manocha 2008], and PDs can be used to determine the existence of a path in planning scenarios [Zhang et al. 2008].

However, it is well-known that much more computation is generally required to determine a PD than a separation distance. For general polyhedral objects, with a total of n faces, the computation time is $O(n^6)$ [Kim et al. 2004b]. This has motivated some researchers to find a more tractable approximation of the PD [Kim et al. 2004a; Kim et al. 2002; Redon and Lin 2006]. Unfortunately, these methods are either too slow for interactive applications [Kim et al. 2004a; Kim et al. 2002] or provide no guaranteed error bound [Redon and Lin 2006].

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, analyze the performance of PolyDepth in different benchmarks, and discuss implementation issues in Sec. 7. We 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 [Cameron and Culley 1986]. If the direction of motion is known, a minimal translational motion in this direction can be found using a multi-resolution mesh hierarchy [Dobkin et al. 1993]. A randomized algorithm was presented by [Agarwal et al. 2000]. 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 [Cameron 1997], expansion of polyhedral approximations [Bergen 2001], and dual-space expansion [Kim et al. 2004b]. 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 [Kim et al. 2002]. Redon and Lin [Redon and Lin 2006] also proposed a CPU/GPU hybrid method of computing a lower bound on the PD, and this algorithm is applicable to polygon-soup models. Lien [Lien 2008; 2009] presented a sampling-based approach to PD computation based on approximate Minkowski sum. More recently, Hachenberger [Hachenberger 2009] 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 [Fisher and Lin 2001]. Moreover, a GPU can be used to accelerate the computation of distance fields [Hoff et al. 2002; Sud et al. 2006]. 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 [Tang et al. 2009]. 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. More recently, a penetration resolution technique based on volume has been presented by [Allard et al. 2010]. This method uses GPUs and can handle deformable objects.

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 interpenetration. 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 [Zhang et al. 2007b; Zhang et al. 2007a]. More recently, kinematical geometry has been used to compute generalized PDs [Nawratil et al. 2009]. Non-Euclidean distance, such as growth distance [Ong and Gilbert 1996; Ong 1993], can also be used as a measure of inter-penetration. Finally, some researchers have investigated penetration volumes instead of PD [Weller and Zachmann 2009], 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 [Dobkin et al. 1993]:

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

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 [Benson 1966;

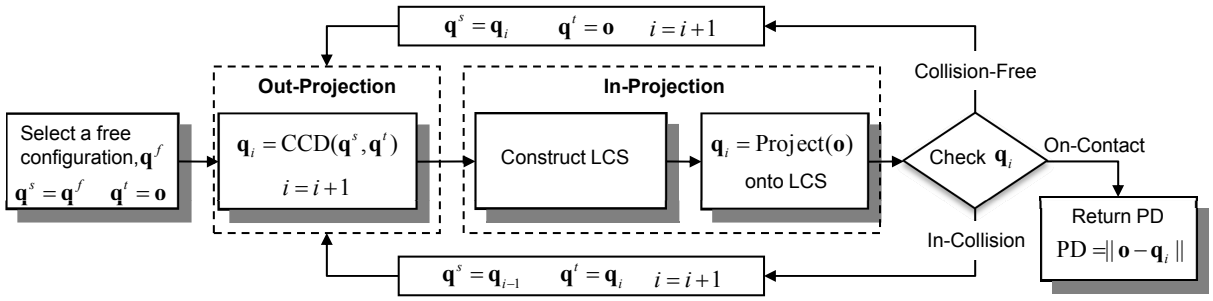


Fig. 1. **PD computation pipeline.**

Cameron 1997] as follows:

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

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

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, $\partial(\mathfrak{A} \oplus -\mathfrak{B})$ [Dobkin et al. 1993]. However, since the interior of the polygon-soup models that we wish to consider may not be closed and thus be undefined, 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 $\mathfrak{A} \oplus -\mathfrak{B}$:

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

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) **Free-configuration selection:** 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. 5).

(2) **Out-projection:** 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). We call the configuration projected on to the contact space the current configuration $\mathbf{q}_i$.

(3) **In-projection:**

  (a) 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.

  (b) 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. 6.1). 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.

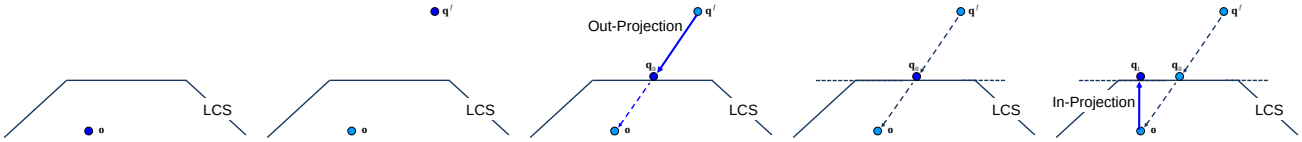
(4) **Sample classification:** since $\mathbf{q}_i$ was obtained from the LCS, and not from the global contact space, $\mathbf{q}_i$ can be in-collision, in-contact or collision-free. We further classify the collision status of $\mathbf{q}_i$ by performing a static collision query on $\mathbf{q}_i$ as 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.

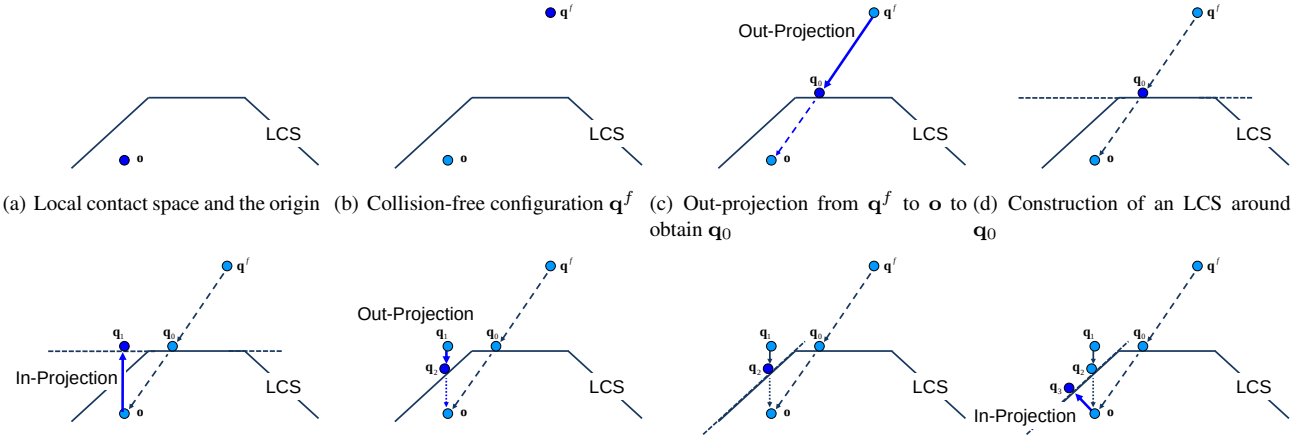
  (c) If $\mathbf{q}_i$ is an in-collision configuration, we obtain a proper contact configuration by setting $\mathbf{q}_i$ to a target configuration ($\mathbf{q}^t \equiv \mathbf{q}_i$), and the previous contact configuration $\mathbf{q}_{i-1}$ becomes the source configuration ($\mathbf{q}^s \equiv \mathbf{q}_{i-1}$). Then we go to step 2.

(5) **Iteration:** steps 2-4 are repeated until the algorithm terminates.



(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 $\mathbf{q}_1$

Fig. 2. **Iterative optimization of a PD for a simple case of convex LCS.** In this simple case, the PD algorithm terminates right after a single iteration of successive out- and in-projections. Then, $PD = ||\mathbf{o} - \mathbf{q}_1||$.



(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 $\mathbf{q}_1$ to $\mathbf{o}$ to obtain $\mathbf{q}_2$ (g) LCS around $\mathbf{q}_2$ (h) In-projection on to the LCS to obtain $\mathbf{q}_3$

Fig. 3. **Iterative optimization of a PD for a slightly complicated case of convex LCS.** In this case, PD requires two iterations since $\mathbf{q}_1$ in (e) corresponds to separation. Then, $PD = ||\mathbf{o} - \mathbf{q}_3||$.

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. More discussion on termination conditions is provided in Sec. 6.3. In Figs. 2 and 3, we illustrate our iterative algorithm for convex LCSs, and in Figs. 10 and 11 for more general cases. These examples illustrate the cases of translational configuration space for two dimensional polygonal objects.

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:

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

There are many methods for CCD, but our choice is conservative advancement (CA) [Lin 1993; Mirtich 1996; Zhang et al. 2006; Zhang et al. 2007c; Tang et al. 2009] 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:

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

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. [Tang et al. 2009] for polygon-soup models. The C^{2}A algorithm uses a bounding volume hierarchy (BVH) based on swept sphere volumes (SSVs) [Larsen et al. 2000] to control the depth of the recursive BVH traversal during iterations of the algorithm, which reduces the computation time sig-

nificantly. Since our PD problem is restricted to translational motion, we can simplify C^2A 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 [Larsen et al. 2000] 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

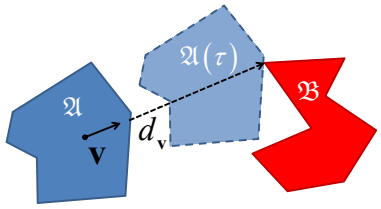


Fig. 4. **Minimal directional distance along $\mathbf{v}$ between two objects.**

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. 4. 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.

[Choi et al. 2006] 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.

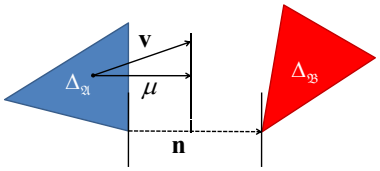


Fig. 5. **Translational CA for triangles.**

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. 5. 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 $\mathfrak{B}$.

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:

$$\mathbf{q}^s = \mathbf{q}^{\mathfrak{A}} + (r_{\mathfrak{A}} + r_{\mathfrak{B}}) \frac{\mathbf{o}^{\mathfrak{A}} - \mathbf{o}^{\mathfrak{B}}}{||\mathbf{o}^{\mathfrak{A}} - \mathbf{o}^{\mathfrak{B}}||},$$

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 $r_{\mathfrak{A}}$ and $r_{\mathfrak{B}}$ are the diameters of spheres enclosing each object. Fig. 6 shows a collision-free configuration determined from the centroid difference.

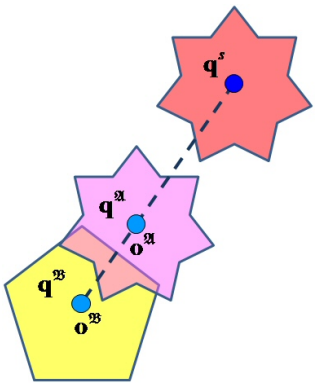


Fig. 6. **A collision-free configuration from the centroid difference.** Obstacle $\mathfrak{B}$ (yellow), initial in-collision configuration of $\mathfrak{A}$ (magenta), and the collision-free configuration of $\mathfrak{A}$ (red).

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

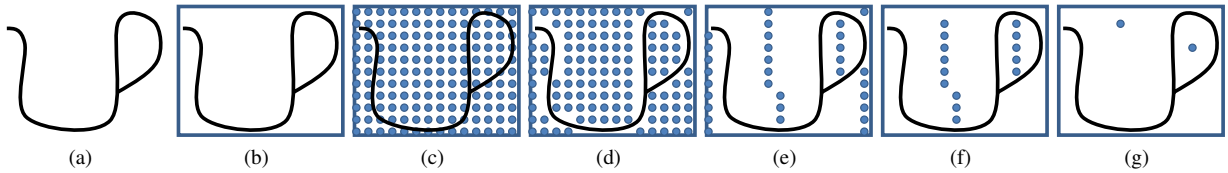


Fig. 7. **Finding maximally clear configurations.** (a) The given model. (b) AABB of the model. (c) Voxelize the AABB and compute the distance to the model from each grid position. (d) Remove the grid points corresponding to small distances. (e) Compare neighboring grid points and remove those with smaller distances by scanning along the x- and y-directions. (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-directions.

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. 8). A similar strategy has been used in workspace sampling in motion planners [Latombe 1991]. Since the construction of a generalized Voronoi diagram for a polygonal model is quite hard [Hoff et al. 1999] we propose a simple algorithm based on a voxelization of space (also see Fig. 7):

(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.

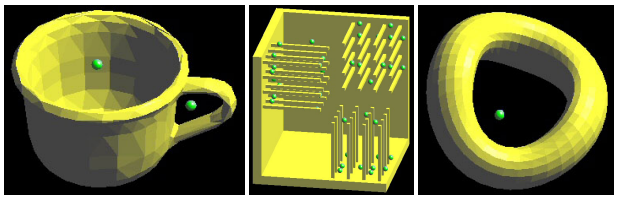


Fig. 8. **Maximally clear configurations for the Cup, Grate, and Distorted-torus models.** The small green spheres show the positions of the maximally clear configurations.

5.3 Motion Coherence

Applications 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. More precisely, in our implementation, we store the last three in-contact configurations obtained from PD computations, and choose the one $\mathbf{q}_c$ that has the minimum translational difference from the input in-collision configuration $\mathbf{o}$. Then, we change the orientation of $\mathbf{q}_c$ to be the same as that of $\mathbf{o}$, and if this does not create any collision, we use $\mathbf{q}_c$ as a starting configuration for the iteration; otherwise, we slightly move $\mathbf{q}_c$ to the opposite direction toward $\mathbf{o}$, and see if it causes collision again. If not, we use $\mathbf{q}_c$ as an initial configuration. Otherwise, we abandon the motion coherence strategy and switch to other methods (e.g. centroid difference).

5.4 Random Configuration

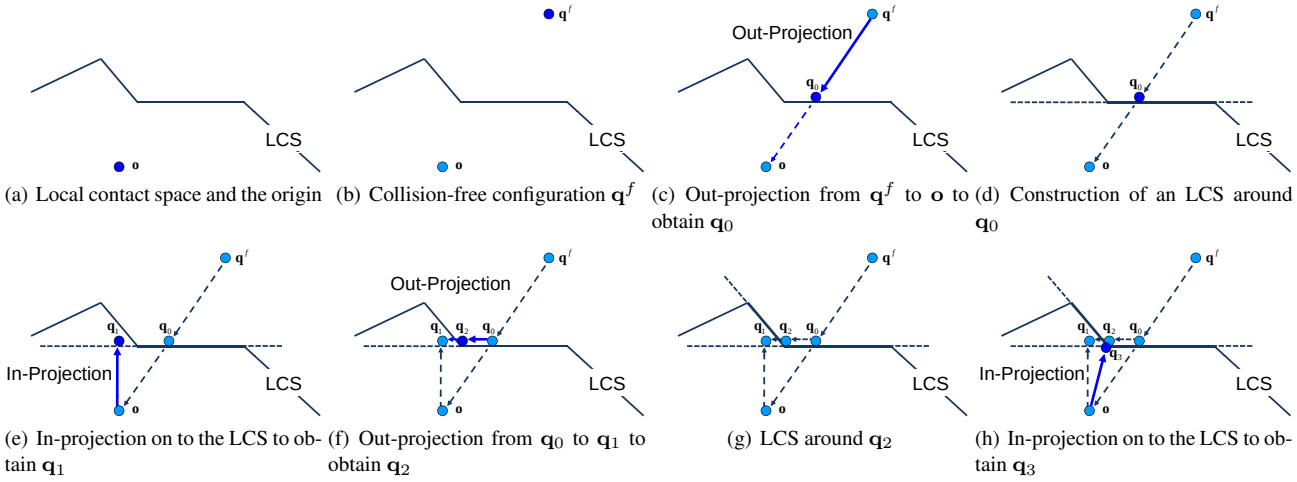
In a more general setting, in which we do not have any knowledge of $\mathbf{q}^s$, we can randomly sample $\mathbf{q}^s$ in the free configuration space [Zhang et al. 2007a] or simply locate $\mathbf{q}^s$ at infinity.

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. 9, this sampling process terminates whenever a collision-free configuration is found. For collision detection, we use [Larsen et al. 2000] in our implementation.

6. ITERATIVE OPTIMIZATION

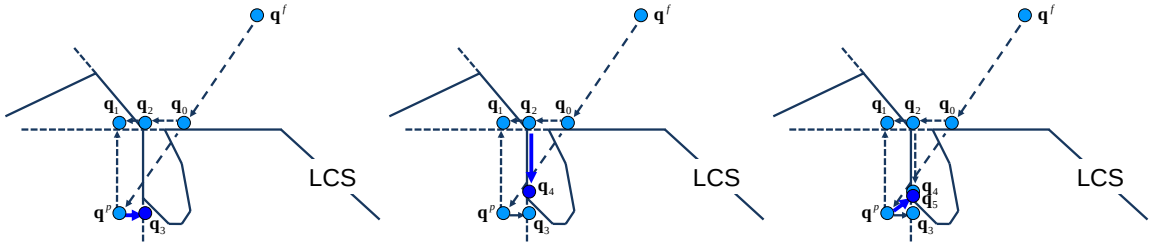
Any sampled configuration $\mathbf{q}$ in the contact space can be used as an estimate of the PD by computing its distance from the origin; i.e. $\mathrm{PD}(\mathfrak{A}, \mathfrak{B}) = \|\mathbf{o} - \mathbf{q}\|$. When we have to deal with a highly non-convex object, finding a good candidate for $\mathbf{q}$ 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.



(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 $\mathbf{q}_3$

Fig. 10. **Iterative optimization of a PD for a non-convex LCS.** The PD is computed after two iterations since $\mathbf{q}_1$ in (e) is in-collision. $PD = ||\mathbf{o} - \mathbf{q}_3||$.



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

Fig. 11. **Iterative optimization of a PD for a more complicated LCS.** Three iterations are required to obtain the PD since $\mathbf{q}_3$ in (a) is in-collision. $PD = ||\mathbf{o} - \mathbf{q}_5||$.

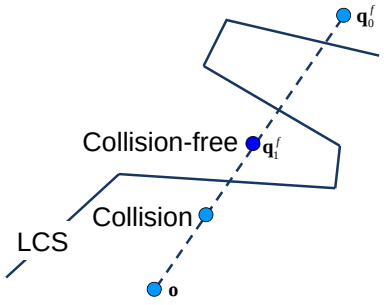


Fig. 9. **Sampling-based search for a collision-free configuration.** The input collision configuration is $\mathbf{o}$ and $\mathbf{q}_0^f$ is an initial collision-free configuration. The first sample on the line from $\mathbf{o}$ to $\mathbf{q}_0^f$ is determined to be in-collision, but the second sample $\mathbf{q}_1^f$ is collision-free.

6.1 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 may degenerate to a lower-dimensional subset of configuration space, such as a line or a point. For example, contact between two collinear edges or a vertex/edge (VE) contact produces a contact line equation, and vertex/vertex (VV) contact produces a point. Our algorithm simply ignores these VV or collinear contacts, 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 [Zhang et al. 2007a]. 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.

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

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

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

It is known [Redon et al. 2002] 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 [Cottle et al. 2009]; however we choose a projected Gauss-Seidel method for simplicity and stability [Jourdan et al. 1998]. Then the LCP reduces to the search for a solution of the following linear system:

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

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 [Golub and Van Loan 1996] 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 the five steps in Sec. 3.2 and Fig.10:

(a) An in-collision configuration (or the origin $\mathbf{o}$) is given as input (Fig. 10(a)).

(b) Select a collision-free configuration $\mathbf{q}^f$ using the technique described in Sec. 5 (step 1 in Sec. 3.2 and Fig. 10(b)).

(c) Using the configurations $\mathbf{q}^f$ as source and $\mathbf{o}$ as target, perform out-projection to obtain a contact configuration $\mathbf{q}_0$ (step 2 in Sec. 3.2 and Fig. 10(c)).

(d) Construct an LCS around the contact configuration $\mathbf{q}_0$ (step 3(a) in Sec. 3.2 and Fig. 10(d)). In this example, the LCS consists of only a single contact plane.

(e) Perform in-projection on to the LCS to obtain $\mathbf{q}_1$ (step 3(b) in Sec. 3.2 and Fig. 10(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 (step 4 in Sec. 3.2).

(f) Again perform out-projection to obtain $\mathbf{q}_2$ (step 2 in Sec. 3.2 and Fig. 10(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 $\mathbf{q}_2$ (step 3(a) in Sec. 3.2 and Fig. 10(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 (step 3(b) in Sec. 3.2 and Fig. 10(h)). The new configuration $\mathbf{q}_3$ is in-contact and the PD that will be returned is $||\mathbf{q}_3 - \mathbf{o}||$ (step 4 in Sec. 3.2).

Fig. 11 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. 11(a)). Thus we perform out-projection to obtain $\mathbf{q}_4$ (Fig. 11(b)), construct the LCS, and run in-projection again to get the final configuration $\mathbf{q}_5$ (Fig. 11(c)).

6.3 Algorithm Termination

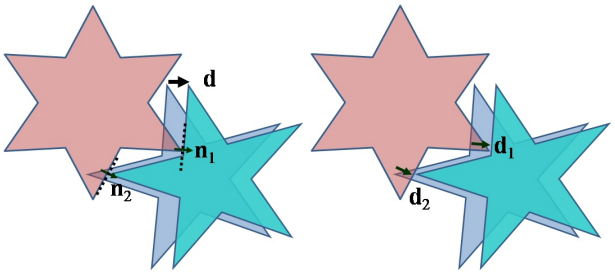
Our algorithm is based on an iterative optimization technique using in- and out-projections. The out-projection ensures that our algorithm maintains an in-contact sample such that a proper upper bound of PD is always obtained at any time of iteration. The in-projection ensures that our algorithm always reduces the PD value during iteration. These projections are iterated until a result from in-projection corresponds to an in-contact configuration so that no further reduction is possible. This way, our algorithm finds a locally-optimal solution for PD.

Since the complexity of the local contact space (LCS) is bounded above by $O(n^6)$ where n is the number of polygons in polygon-soup models, the number of iterative projections in our algorithm is finite. Moreover, since our algorithm reduces a PD value at every iteration, no projection can be repeated for the same LCS. Therefore, our algorithm always terminates after a finite number of steps, which is typically 2 ˜ 3 in practice.

6.4 Estimation of Local Penetration Depth

When two objects intersect 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 [Guendelman et al. 2003; Tang et al. 2009], is required to compute torques or to stabilize the simulation. [Guendelman et al. 2003] compute local PDs from distance fields, whereas [Tang et al. 2009] 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



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

Fig. 12. **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 $\mathbf{d}_2$.

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. In other words, the locally deepest penetrating points are the ones that would be resolved later than any other points; thus after the resolution by global PD, these points will just touch each other. Since PD allows only rigid translational motion, the amount of motion to resolve deepest penetrating points is obtained by projecting the PD translation onto the penetrating direction of the penetrating points, which is identical to the normal of LCS corresponding to these points.

More specifically, we compute the local PD $\mathbf{d}_i$ of the ith interpenetrating region as follows (also see Fig. 12):

(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 [Guendelman et al. 2003; Tang et al. 2009], 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 [Guendelman et al. 2003], and no expensive Boolean intersection is required unlike [Tang et al. 2009]. Also note that, according to our definition of local PD, even if $\mathbf{d}$ is not zero but if $\mathbf{d}$ is normal to $\mathbf{n}$, the local PD becomes zero. This is the case where the corresponding contact features are sliding.

6.5 Computational Complexity of Algorithm

Our algorithms consist of two main procedures: out- and in-projections. In our algorithm, the out-projection is implemented using translational CCD. The computational complexity of translational CCD is similar to that of distance computation using bounding volume hierarchies (BVHs), since our translational CCD is equivalent to computing the minimal directional distance (MDD) between BVHs. According to [Larsen et al. 2000], the cost function T for distance calculation can be analyzed as $T = N_{bv} \times C_{bv} + N_p \times C_p$, where N_{bv} is the number of bounding volume pair operations, C_{bv} is the cost for pairwise distance computation for BVs, N_p is the number of primitive pair operations, and C_p is the cost for pairwise distance computation for primitives. Translational CCD has a similar cost function, but C_{bv} and C_p now correspond to the costs to compute the MDD for BVs and primitives, respectively. Moreover, we use an iterative method to calculate MDD, and each iteration requires Euclidean minimum distance calculation between a BV or triangle pair, which is a constant. In practice, the average number of iterations for MDD calculation is a small constant; for example, for two bunny models with 40K triangles each, as shown in Fig. 13, it is 1.7. This means that the computational cost for translation CCD is only 1.7 times the cost for a minimum distance query (the function T above).

The main computational cost for in-projection lies in solving Eq. 11. Since $\mathbf{J}\mathbf{J}^T$ is an $n \times n$ matrix where n is the number of contact features returned by out-projection, solving the linear system takes $O(n^3)$ time, but the Gauss-Seidel iterative solver would take $O(N_G n^2)$ time where N_G is the number of Gauss-Seidel iterations. Thus, the total computational complexity of our algorithm is $(T + O(N_G n^2))N$ where N is the total number of iterations for in- and out-projections.

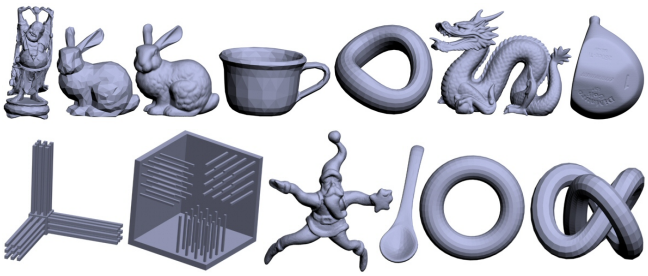


Fig. 13. **Benchmarking models with triangle counts. Top row:** Buddha (10K), Bunny1 (1K), Bunny2 (40K), Cup (1K), Distorted-torus (1.33K), Dragon (174K), Golf-club (105K) **Bottom row:** Grate1 (0.54K), Grate2 (0.94K), Santa (152K), Spoon (1.34K), Torus (2K), and Torusknot (3K).

7. RESULTS AND DISCUSSION

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

7.1 Implementation and Benchmarks

We implemented our PD computation algorithm[1] 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. 13. 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.

[1]The source codes are available for download at `http://graphics.ewha.ac.kr/PolyDepth`

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. 14 shows an example in which two copies of the Torusknot model intersect.

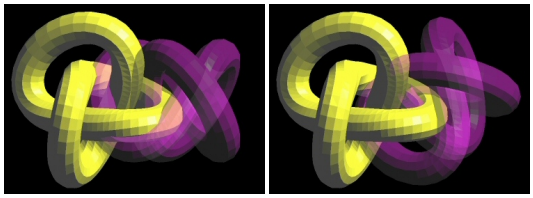


Fig. 14. **Two Torusknot models in a random configuration.**

We use a centroid difference to find the initial collision-free configuration. The performance of our algorithm is characterized in Table I. 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. 15, we also show the number of contact feature pairs, number of iterations, and computation time for two intersecting Torusknots in random configurations.

Table I. **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

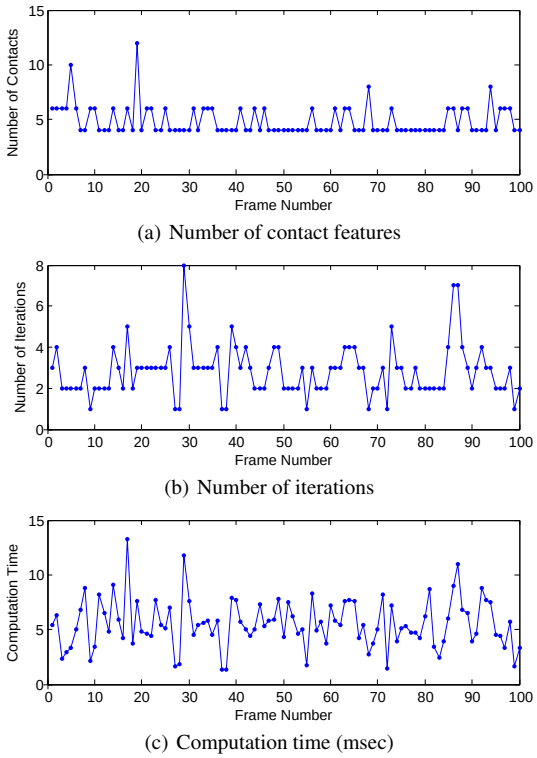
For the more topologically challenging models such as Grate1, Grate2 and Distorted-torus, we use maximally clear configurations and sampling-based search scheme to find the initial collision-free configurations, because of their very concave geometries. Results are shown in Fig. 16.

In order to assess the accuracy of our PD algorithm, we used Lien's open source library [Lien 2009] 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. 17 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.



(a) Number of contact features

(b) Number of iterations

(c) Computation time (msec)

Fig. 15. **Graphs of PD computation for two Torusknot models in 100 random configurations.**

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 II.

Table II. **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 Havok^{TM2}, a rigid body dynamics simulator, we generated paths for the Dragon (Fig. 18), and Spoon and Cup models (Fig. 19), and tested our PD algorithm along these paths. Fig. 20 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. 18. The centroid difference explained in Sec. 5.1 is used to find an initial collision-free configuration.

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

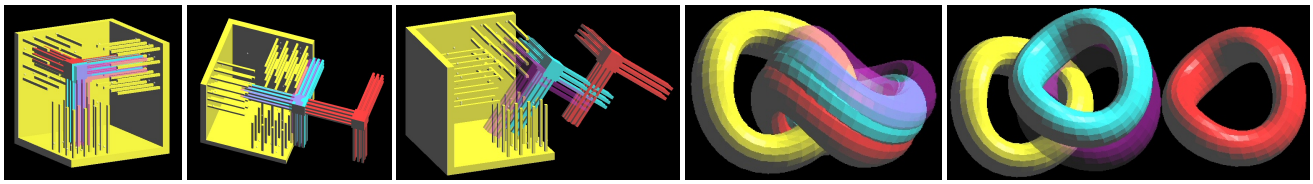
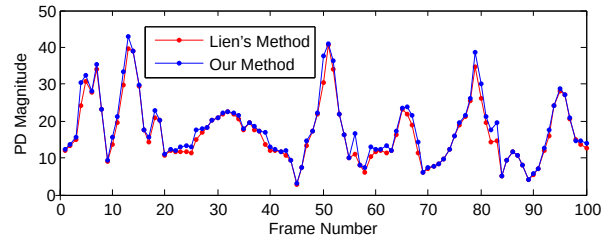
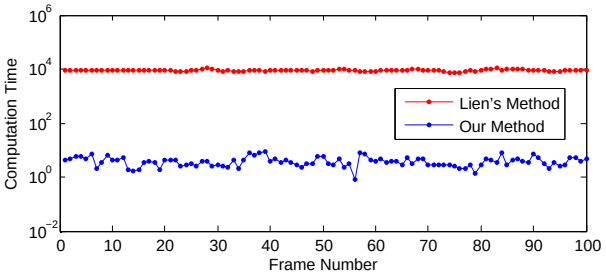


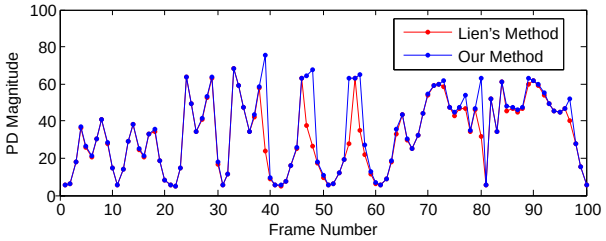
Fig. 16. **Using maximally clear configurations and sampling-based search to determine collision-free configurations for the Grate and Distorted-torus models.** Obstacle $\mathfrak{B}$ is yellow, and the input in-collision configuration of $\mathfrak{A}$ is semitransparent magenta. The initial collision-free configuration of $\mathfrak{A}$ is red, and the configuration of $\mathfrak{A}$ translated by PD is shown in cyan.



(a) PD Magnitude (Bunny1)



(b) Computational Time (Bunny1)



(c) PD Magnitude (Distorted Torus)

Fig. 17. **Graphs of PD comparisons using our method and Lien's method [Lien 2009] at random configurations.**

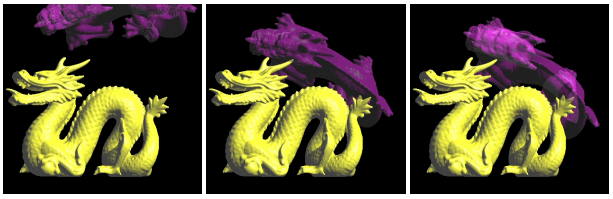
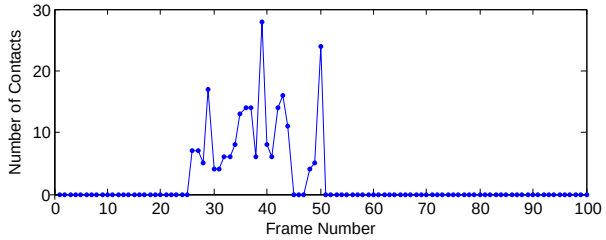
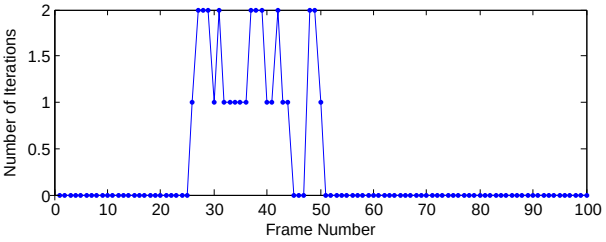


Fig. 18. **Path for the Dragon model generated by a physics simulator.**

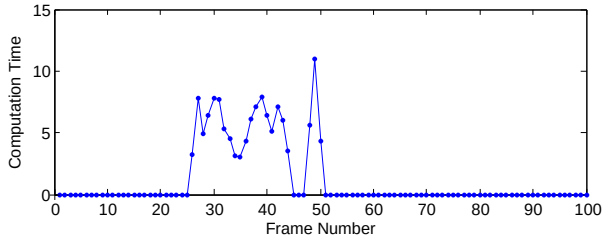
Fig. 19. **Path of the Spoon model relative to Cup generated by a physics simulator.**



(a) Number of Contact Features



(b) Number of Iterations

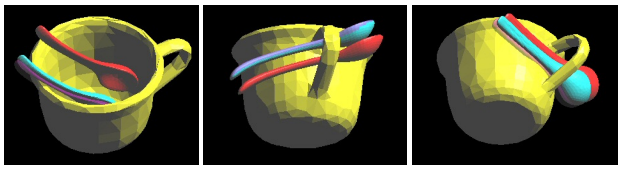


(c) Computation Time (msec)

Fig. 20. **Graphs of PD computation using two colliding Dragon models moving on predefined paths.**

Fig. 22 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. 21 shows the results when maximally clear configurations were used for the same scene. Fig. 8(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. 22(b). But in the same situations, the maximally clear configurations are very satisfactory, as demonstrated in Fig. 22(e)∼(j). The performance of our algorithm for predefined paths is summarized in Table III.



(a) (b) (c)

Fig. 21. **PD results using maximally clear configurations for Spoon and Cup models moving on predefined paths.**

Table III. **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

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. 23). We used impulse-based dynamics [Guendelman et al. 2003] to simulate rigid-body dynamics based on the local PD method presented in Sec. 6.4. 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 IV. Note that additional time is required for local PD computations.

Table IV. **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

7.2 Discussion

There are some limitations to our algorithm. An object with high geometric and topological complexities may require a large number of iterations to have a convergent solution. Furthermore, our method only approximates an upper bound, which can depend on the initial collision-free configuration, as Fig. 22(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:

(1) Maximally clear configurations are suitable for strongly non-convex objects of high genus, such as the Grate, Torus and Cup in Figs. 16 and 21, since a collision-free configuration can be located inside a concavity.

(2) Sampling-based search is useful for intricate or interlocking objects such as those shown in Fig. 16, but it is time-consuming.

(3) In a dynamics simulation, such as that shown in Fig. 23, 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.

(4) 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.

(5) If the preprocessing cost of finding a maximally clear configurations is unaffordable, and no application-dependent information is available, random sampling can be employed.

In order to automate the selection of an initial configuration, we suggest following the above sequence and choose the resulting configuration closest to the input. However, 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.

7.3 Comparisons against Recent Approaches

In this section, we make qualitative comparisons of our algorithm against recent algorithms related to penetration depth computation including [Nawratil et al. 2009; Allard et al. 2010].

Nawratil et al.'s method [Nawratil et al. 2009] iteratively calculates generalized penetration depth using kinematical geometry and constrained optimization. However, their method has the following differences compared to ours:

—Since the goal of their work is to compute a minimal rigid body displacement to separate overlapped objects by using a kinematical mapping in $\mathbb{R}^{12}$, their computational complexity is much higher than ours both in theory and in practice. It is unclear whether the application of this method to translational penetration depth computation can perform at interactive rates like ours.

—They also suggest two methods for finding an initial collision-free configuration, but this procedure takes a considerable amount of time to perform, e.g. a second for a model consisting of a few thousands triangles, which is too expensive for interactive applications.

Allard et al.'s work is also related to penetration resolution [Allard et al. 2010]. However, their method is different from ours in the following sense:

—Their work on penetration resolution is based on dynamics simulation, whereas ours is based on purely geometric formulation, and they do not have any guarantees on collision resolution, whereas ours does.

—Their work has no sense of optimality for collision resolution, whereas ours does.

—Their work may suffer from the image resolution imposed by the use of GPUs, but ours does not.

—Their work is suitable for deformable objects as well as rigid ones, whereas ours is suitable for rigid objects.

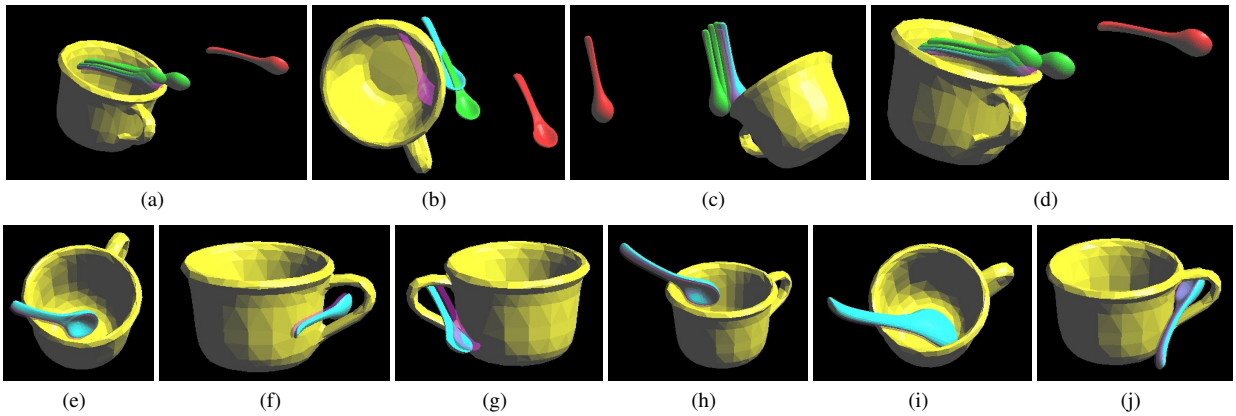


Fig. 22. **PD results for the Spoon and Cup models moving on predefined paths.** Obstacles are shown in yellow, input in-collision configuration in semi-transparent magenta, initial collision-free configurations in red, contact configurations from in- and out-projections in green, and the configuration 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.

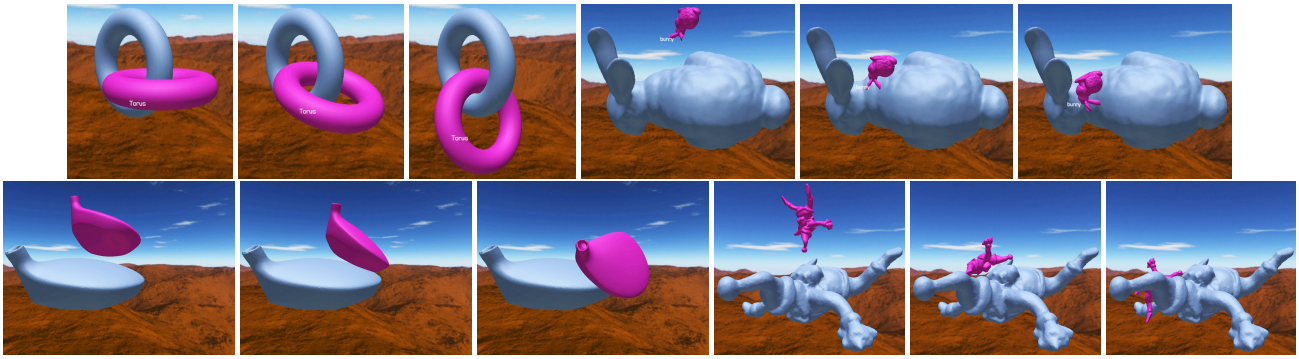


Fig. 23. **Dynamic simulations results of pairs of Torus, Bunny2, Golf-club and Santa models using impulse-based dynamics [Guendelman et al. 2003].**

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 generalized PD problem [Zhang et al. 2007b; Zhang et al. 2007a; Nawratil et al. 2009] in which both translational and rotational motions are possible. The key to making this problem 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.

ACKNOWLEDGMENTS

This research work was supported in part by the NRF grant funded by the Korea government (MEST) (No. 2009-0086684) and IT R&D program of MKE/MCST/KOCCA (2008-F-033-02).

REFERENCES

AGARWAL, P. K., 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.

ALLARD, J., FAURE, F., COURTECUISSE, H., FALIPOU, F., DURIEZ, C., AND KRY, P. G. 2010. Volume contact constraints at arbitrary resolution. *ACM Trans. Graph. 29*, 82:1–82:10.

BENSON, R. V. 1966. *Euclidean Geometry and Convexity*. McGraw-Hill, New York, NY.

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

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

CAMERON, S. A. AND CULLEY, R. K. 1986. Determining the minimum translational distance between two convex polyhedra. In *Proc. of IEEE Inter. Conf. on Robotics and Automation*. 591–596.

CHOI, Y.-K., LI, X., RONG, F., WANG, W., AND CAMERON, S. 2006. Computing the minimum directional distance between two convex polyhedra. *HKU CS Tech Report TR-2006-01*.

COTTLE, R., PANG, J., AND STONE, R. 2009. *The linear complementarity problem*. Society for Industrial & Applied.

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

FISHER, S. AND LIN, M. C. 2001. Fast penetration depth estimation for elastic bodies using deformed distance fields. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 2001*.

GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. 22,* 3, 871–878.

HACHENBERGER, P. 2009. Exact Minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica 55,* 2, 329–345.

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

HOFF, K., ZAFERAKIS, A., LIN, M., AND MANOCHA, D. 2002. Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Tech. Rep. TR02-004, Department of Computer Science, University of North Carolina.

JOURDAN, F., ALART, P., AND JEAN, M. 1998. A Gauss-Seidel like algorithm to solve frictional contact problems. *Computer Methods in Applied Mechanics and Engineering 155,* 1-2, 31–47.

KIM, Y., LIN, M., AND MANOCHA, D. 2004a. Fast penetration depth estimation using rasterization hardware and hierarchical refinement. In *Algorithmic Foundations of Robotics V*. Springer Tracts in Advanced Robotics, vol. 7. Springer Berlin / Heidelberg, 505–522.

KIM, Y. J., LIN, M., AND MANOCHA, D. 2004b. Incremental penetration depth estimation between convex polytopes using dual-space expansion. *IEEE Trans. on Visualization and Computer Graphics 10,* 1, 152–164.

KIM, Y. J., OTADUY, M. A., LIN, M. C., AND MANOCHA, D. 2002. Fast penetration depth computation for physically-based animation. *Proc. of ACM Symposium on Computer Animation*.

KIM, Y. J., OTADUY, M. A., LIN, M. C., AND MANOCHA, D. 2003. Six-degree-of-freedom haptic rendering using incremental and localized computations. *Presence 12,* 3, 277–295.

LARSEN, E., GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 2000. Fast distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation*, 3719–3726.

LATOMBE, J.-C. 1991. *Robot Motion Planning*. Kluwer Academic Publishers, Boston.

LIEN, J.-M. 2008. Covering Minkowski sum boundary using points with applications. *Computer Aided Geometric Design 25,* 8 (November), 652–666.

LIEN, J.-M. 2009. A simple method for computing Minkowski sum boundary in 3d using collision detection. In *Algorithmic Foundation of Robotics VIII*, G. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds. Springer Tracts in Advanced Robotics, vol. 57. Springer Berlin / Heidelberg, 401–415.

LIN, M. AND MANOCHA, D. 2003. Collision and proximity queries. *Handbook of discrete and computational geometry*, 787–807.

LIN, M. C. 1993. Efficient collision detection for animation and robotics. Ph.D. thesis, University of California, Berkeley, CA.

MIRTICH, B. V. 1996. Impulse-based dynamic simulation of rigid body systems. Ph.D. thesis.

NAWRATIL, G., POTTMANN, H., AND RAVANI, B. 2009. Generalized penetration depth computation based on kinematical geometry. *Comput. Aided Geom. Des. 26*, 425–443.

ONG, C. 1993. Penetration distances and their applications to path planning. Ph.D. thesis, Michigan Univ., Ann Arbor.

ONG, C. J. AND GILBERT, E. 1996. Growth distances: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation 12,* 6, 888–903.

REDON, S. 2004. Fast continuous collision detection and handling for desktop virtual prototyping. *Virtual Reality 8*, 63–70.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Gauss' least constraints principle and rigid body simulations. In *In proceedings of IEEE International Conference on Robotics and Automation*. 11–15.

REDON, S. AND LIN, M. C. 2006. A fast method for local penetration depth computation. *journal of graphics tools 11,* 2, 37–50.

SUD, A., GOVINDARAJU, N., GAYLE, R., KABUL, I., AND MANOCHA, D. 2006. Fast proximity computation among deformable models using discrete Voronoi diagrams. In *ACM SIGGRAPH*. ACM New York, NY, USA, 1144–1153.

TANG, M., KIM, Y. J., AND MANOCHA, D. 2009. C^2A: controlled conservative advancement for continuous collision detection of polygonal models. In *IEEE International Conference on Robotics and Automation*. 356–361.

TANG, M., LEE, M., AND KIM, Y. J. 2009. Interactive Hausdorff distance computation for general polygonal models. *ACM Trans. Graph. 28*, 74:1–74:9.

WELLER, R. AND ZACHMANN, G. 2009. Inner sphere trees for proximity and penetration queries. In *Proceedings of Robotics: Science and Systems*. Seattle, USA.

ZHANG, L., KIM, Y., AND MANOCHA, D. 2007a. A fast and practical algorithm for generalized penetration depth computation. In *Proceedings of Robotics: Science and Systems*. Atlanta, USA.

ZHANG, L., KIM, Y., AND MANOCHA, D. 2008. Efficient cell labelling and path non-existence computation using c-obstacle query. *The International Journal of Robotics Research 27,* 11-12 (Nov-Dec), 1246–1257.

ZHANG, L., KIM, Y., VARADHAN, G., AND MANOCHA, D. 2007b. Generalized penetration depth computation. *Computer-Aided Design 39,* 8, 625–638.

ZHANG, L. AND MANOCHA, D. 2008. An efficient retraction-based RRT planner. In *IEEE International Conference on Robotics and Automation (ICRA)*. 3743–3750.

ZHANG, X., LEE, M., AND KIM, Y. J. 2006. Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer 22*, 749–760.

ZHANG, X., REDON, S., LEE, M., AND KIM, Y. J. 2007c. Continuous collision detection for articulated models using Taylor models and temporal culling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) 26,* 3, 15.