

Generalized Penetration Depth Computation

Liangjun Zhang¹ Young J. Kim² Gokul Varadhan¹ Dinesh Manocha¹

¹ Dept. of Computer Science, University of North Carolina at Chapel Hill, USA, {zlj,varadhan,dm}@cs.unc.edu

² Dept. of Computer Science and Engineering, Ewha Womans University, Korea, kimy@ewha.ac.kr

<http://gamma.cs.unc.edu/PDG>

Abstract

Penetration depth (PD) is a distance metric that is used to describe the extent of overlap between two intersecting objects. Most of the prior work in PD computation has been restricted to *translational PD*, which is defined as the minimal translational motion that one of the overlapping objects must undergo in order to make the two objects disjoint. In this paper, we extend the notion of PD to take into account both translational and rotational motion to separate the intersecting objects, namely *generalized PD*. When an object undergoes rigid transformation, some point on the object traces the longest trajectory. The generalized PD between two overlapping objects is defined as the minimum of the longest trajectories of one object under all possible rigid transformations to separate the overlapping objects.

We present three new results to compute generalized PD between polyhedral models. First, we show that for two overlapping convex polytopes, the generalized PD is same as the translational PD. Second, when the complement of one of the objects is convex, we pose the generalized PD computation as a variant of the convex containment problem and compute an upper bound using optimization techniques. Finally, when both the objects are non-convex, we treat them as a combination of the above two cases, and present an algorithm that computes a lower and an upper bound on generalized PD. We highlight the performance of our algorithms on different models that undergo rigid motion in the 6-dimensional configuration space. Moreover, we utilize our algorithm for complete motion planning of polygonal robots undergoing translational and rotational motion in a plane. In particular, we use generalized PD computation for checking path non-existence.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

Keywords: Penetration depth

1 Introduction

Calculating a distance measure is a fundamental problem that arises in many applications such as physically-based modeling, robot motion planning, virtual reality, haptic rendering and computer games. Typical distance measures employed in these applications include separation distance, Hausdorff distance, spanning distance and penetration depth [Lin and Manocha 2003]. Among these measures, penetration depth (PD) is used to quantify the extent of interpenetration between two overlapping, closed, geometric objects.

PD computation is important in a number of applications. In rigid body dynamics, inter-penetration between simulated objects is often unavoidable due to the nature of discrete, numerical simulation. As a result, several response algorithms like penalty-based simulation methods need the PD information to compute the non-penetration constraint force [Mirtich 2000; Stewart and Trinkle 1996]. The PD is also used to estimate the time of contact to apply impulsive forces in impulse-based methods [Kim et al. 2002a]. Sampling-based motion planning techniques perform PD computation between the robot and the obstacles to generate samples in narrow passages in the configuration space [Hsu et al. 1998]. Many 6-DOF haptic rendering algorithms use penalty-based methods to compute a collision response and need to compute the PD at haptic update rates [Kim et al. 2003]. Other applications include tolerance verification, where PD could be used to estimate the extent of interference between the parts of a machine structure [Requicha 1993].

Most of the prior work on PD computation has been restricted to *translational PD*. The translational PD between two overlapping objects is often defined as the minimum translational distance needed to separate the two objects. Many good algorithms to estimate the translational PD between convex and non-convex polyhedra are known [van den Bergen 2001; Kim et al. 2002b; Kim et al. 2002a]. However, translational PD computation is not sufficient for many applications as it does not take into account the rotational motion. For example, in rigid body dynamics simulations, objects undergo both translational and rotational motion due to external forces and torques. In order to compute an accurate collision response, we also need to take into account rotational motion during PD computation. Similarly in 6-DOF haptic rendering, the rotational component in penalty forces, such as torque, should be considered in order to compute the response force. Also, since the configuration space of a rigid polyhedral model is a 6-dimensional space, the rotational PD metric is important for motion planning.

In this paper, we take into account the translational and rotational motion to describe the extent of two intersecting objects and refer to that extent of inter-penetration as the *generalized penetration depth*. When an object undergoes rigid transformation, some point on the object traces the longest trajectory. The generalized PD between two overlapping objects is defined as the minimum of the longest trajectories of one object under all possible rigid transformations to separate the overlapping objects. To the best of our knowledge, there is no prior work on generalized PD computation between polyhedral models.

In general, computing the generalized PD between two non-convex polyhedra is more difficult than computing translational PD due to the non-linear rotational term embedded in the definition. In case of translational PD, the problem reduces to computing the closest point from the origin to the boundary of the Minkowski sum of the primitives. The combinatorial complexity of Minkowski sum can be as high as $O(m^3n^3)$ for non-convex polyhedra, where m and n are the number of features in the two polyhedra. However, no similar formulation is known to compute the generalized PD. Computing generalized PD can be viewed as minimizing a distance metric in the configuration space. The configuration space for the case of generalized PD computation in 3D is 6-dimensional and the problem reduces to computing an arrangement of $O(n^2)$ five-

dimensional contact hyper-surfaces. The combinatorial complexity of the arrangement is $O(n^{12})$ [Halperin 2005].

1.1 Main Results

We present a formulation of generalized PD and present novel results related to computing PD between polyhedral models. These include:

- We propose a novel definition for generalized penetration depth to quantify both the translational and rotational amount of inter-penetration between two overlapping polyhedra.
- We prove that for convex models, their generalized PD is the same as translational PD.
- We present an approximate algorithm to compute the generalized PD for non-convex models. We compute a lower bound on the generalized PD by using convex-covering techniques and computing the translational PD for each pair of convex polytopes.
- We reduce the problem of computing an upper bound for the generalized PD to a variant of 3D convex containment problem using linear programming.

We have implemented our algorithm and applied to many non-convex 3D models undergoing rigid motion in 6-dimensional configuration space. The running time varies based on model complexity and the relative configuration of two objects. In practice, our algorithm takes about 2 ms to 6 ms on 2.8 GHz PC to compute the lower bound on generalized PD, and 21 ms to 1.02 sec for the upper bound. We also use our algorithms to perform *C-obstacle query* for complete and collision-free motion planning of planar robots. We use this query as part of a sampling-based complete motion planning algorithm and use the generalized PD computation to accelerate the check for path non-existence.

1.2 Organization

The rest of the paper is organized in the following manner. Section 2 briefly surveys the previous work on PD computation and distance metrics in configuration space. Section 3 presents our formulation of generalized PD and highlights many of its properties. In Section 4, we show that the generalized PD computation between convex polytopes is same as translational PD computation. Section 5 highlights the relationship between generalized PD computation and the containment problem, which enables us to estimate the generalized PD between non-convex models in Section 6. Section 7 and 8 present our experimental results and an application to complete motion planning.

2 Previous Work

In this section, we give a brief overview of prior work on PD computation and distance metrics in configuration space (C-space).

2.1 PD and Arrangement Computation

Given a finite set of hypersurfaces \mathcal{S} in R^d , their arrangement $\mathcal{A}(\mathcal{S})$ is the decomposition of R^d into cells \mathcal{C} of dimensions $0, 1, \dots, d$. Here, a k -dimensional cell \mathcal{C}^k in $\mathcal{A}(\mathcal{S})$ is a maximal connected set contained in the intersection of a subset of the hypersurfaces in \mathcal{S} that is not intersected by any other hypersurfaces in \mathcal{S} [Halperin 1997]. It is well known that the worst case combinatorial complexity of an arrangement of n hypersurfaces in R^d is $O(n^d)$.

Since both the translational and generalized PD can be formulated in C-space, the complexity of computing both the PD's is governed by that of C-space boundary. In case of polyhedral objects in 3D, their C-space can be computed by enumerating their contact surfaces and computing their arrangement [Latombe 1991]. As a result, one can calculate the PDs by computing the arrangement of contact surfaces. However, the combinatorial complexity of the arrangement is $O(n^{12})$ [Halperin 2005]. Moreover, in practice, robust computation of arrangements is known to be a hard problem [Raab 1999].

2.2 Translational Penetration Depth

The translational PD, PD^t , is defined as a minimum translational distance to make two objects disjoint. This definition can be formulated in terms of the *Minkowski sum* of two objects [Dobkin et al. 1993]. Several algorithms have been proposed for exact or approximate computation of PD^t . Bergen proposes a quick lower bound estimation to PD^t between two convex polytopes by iteratively expanding a polyhedral approximation of the *Minkowski sum* [van den Bergen 2001]. Kim et al. [2002b] presents an incremental algorithm to estimate a tight upper bound on PD^t between convex polytopes by walking to a “locally optimal solution”. They have also presented an algorithm to compute an approximation of global PD^t between two general polyhedral models by using hierarchical refinement [2002a]. The hierarchical refinement approach decomposes the non-convex objects into convex polytopes and uses a bounding volume hierarchy to recursively refine the estimation of PD^t . Redon et al. [2005] describe a fast method to compute an approximation of the local penetration depth between two general polyhedral models using graphics hardware. The best known theoretical algorithm to compute PD^t between convex polytopes is given in [Agarwal et al. 2000] and its running time is $O(m^{\frac{3}{4}+\epsilon}n^{\frac{3}{4}+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon})$ for any positive constant ϵ , where m and n denote the number of features in the two polytopes. However, we are not aware of any implementation of this algorithm. In case of general polyhedral models, it is known that the computational complexity of PD^t computation can be as high as $O(m^3n^3)$ [Kim et al. 2002a].

2.3 Generalized Penetration Depth

To the best of our knowledge, there is no prior published work on generalized PD computation for either convex or non-convex polyhedral objects. If we view the problem of separating the object A from B as placing A into \bar{B} - the complement space of B , the most closely related work to generalized PD is the 2D polygon containment algorithms [Chazelle 1983; Milenkovic 1999; Grinde and Cavalier 1996; Avnaim and Boissonnat 1989; Agarwal et al. 1998] and rotational overlapping minimization [Milenkovic 1998; Milenkovic and Schmidl 2001].

The standard 2D polygon containment problem is to check whether a polygon Q with n vertices can contain another polygon P with m vertices. For general non-convex polygons, the time complexity of this problem is $O(m^3n^3 \log(mn))$ [Avnaim and Boissonnat 1989]. When restricted to convex objects, the time complexity of the 2D containment problem can be significantly improved. [Chazelle 1983] proposed an enumerative algorithm with an $O(mn^2)$ time complexity. [Milenkovic 1999; Grinde and Cavalier 1996] used mathematical programming techniques to compute an optimal solution.

Given an overlapping layout of polygons inside a container polygon, the rotational overlapping minimization problem is to compute the translation and rotation motion to minimize their overlap. [Milenkovic 1998] pose this problem as constraint-solving and

employ mathematical programming methods to solve it. By using the non-overlapping property as a hard constraint, [Milenkovic and Schmidl 2001] minimize a quadratic function of the position and orientation of objects to compute a non-overlapping layout based on quadratic programming.

2.4 Distance Metric in Configuration Space

The configuration space of an object is the space for all possible placements of this object in environment [Latombe 1991; LaValle 2006]. For example, if a rigid object in 3D can translate and rotate, its C-space is 6-dimensional. A configuration is called *free* if the placement of the object at that configuration does not result in a collision with other obstacles in the environment. Otherwise, it is a *colliding* configuration. Essentially, the PD is a distance metric in C-space to represent a shortest distance from a given, colliding configuration to all the free configurations.

When only translation is allowed in the distance metric, the corresponding configuration space can be formulated using the Minkowski sum, which has $O(n^2)$ combinatorial complexity for two convex polytopes (with n features), and $O(n^6)$ for non-convex polyhedra [Halperin 2002]. Since only translation is allowed, we can use the Euclidean distance between two configurations as a distance metric. Therefore, the translational penetration depth computation, which finds a nearest point on the surface of Minkowski sum to the origin, has the same combinatorial complexity as the Minkowski sum formulation.

When both translation and rotation are allowed (i.e., 6-DOF C-space), the corresponding C-space becomes more complex and its combinatorial complexity is $O(n^{12})$ for 3D non-convex polyhedra [Halperin 2005]. Moreover, in 6DOF C-space, it is difficult to define a meaningful distance metric that can encode both translational and rotational movement than the one in 3-DOF, Euclidean space [Kuffner 2004; Amato et al. 2000].

The L_p ($p \geq 1$) metric is one of the important family of metrics in 6DOF C-space [LaValle 2006]. Another important distance metric is the displacement metric; this is the minimum Euclidean displacement distance between all the points on the model when it is at two different configurations.[LaValle 2006].

3 Generalized Penetration Depth

The translational PD, PD^t is defined as a minimum translation distance to separate two overlapping objects A and B :

$$PD^t(A, B) = \min(\{\|\mathbf{d}\| \mid \text{interior}(A + \mathbf{d}) \cap B = \emptyset\}), \quad \mathbf{d} \in \mathcal{R}^3. \quad (1)$$

In our work, we extend the notion of PD^t by taking into account translational as well as rotational motion to separate the overlapping objects. Before proceeding to the definition of generalized PD, we first introduce our notation that is used throughout this paper.

3.1 Notation

We use a bold face letter, such as the origin \mathbf{o} , to distinguish a vector quantity from a scalar quantity. We use a sextuple $(x, y, z, \phi, \theta, \psi)$ to encode the 6-dimensional *configuration* of a 3D object, where x, y and z represent the translational components, and ϕ, θ and ψ are an Euler angle representation for the rotational components. The rotation component can be also represented as a rotation vector $r = (r_1, r_2, r_3)^T = \alpha \hat{\mathbf{a}}$, where α is the rotation angle and $\hat{\mathbf{a}}$ is the rotation vector. $A(\mathbf{q})$ is a placement of an object A at configuration \mathbf{q} , and $\mathbf{p}(\mathbf{q})$ is the corresponding position of a point \mathbf{p} on A .

3.2 Distance Metric D_g In C-Space

In order to define generalized PD, PD^g , we first introduce a distance metric D_g defined in configuration space (or C-space). We use this metric to measure the distance of an object A at two different configurations.

Let l_t be a curve in C-space, which connects two configurations \mathbf{q}_0 and \mathbf{q}_1 (Fig. 1-(a)) and is parameterized in t . When the configuration of A changes along the curve l , any point \mathbf{p} on A will trace out a trajectory in 3D Euclidean space shown in Fig. 1-(b). This trajectory can be represented as $r = \mathbf{p}(l(t))$, and its arc-length $\mu(\mathbf{p}, l)$, which is denoted as *trajectory length*, can be calculated as:

$$\mu(\mathbf{p}, l) = \int \|\dot{\mathbf{p}}(l(t))\| d(l(t)).$$

As Fig. 1-(a) shows, there can be multiple curves connecting two configurations \mathbf{q}_0 and \mathbf{q}_1 . When A moves along any such curve, some point on A corresponds to the longest *trajectory length* as compared all other points on A . For each C-space curve connecting \mathbf{q}_0 and \mathbf{q}_1 , we consider the corresponding longest *trajectory length*. We define the distance metric $D_g(\mathbf{q}_0, \mathbf{q}_1)$ as the minimum over all longest *trajectory lengths* (Fig. 1-(c)):

$$D_g(\mathbf{q}_0, \mathbf{q}_1) = \min(\{\max(\{\mu(\mathbf{p}, l) \mid \mathbf{p} \in A\}) \mid l \in L\}), \quad (2)$$

where L is a set of all the curves connecting \mathbf{q}_0 and \mathbf{q}_1 .

Properties of D_g metric. The distance metric defined above has the following properties [LaValle 2006]:

- **Non-negativity:** $D_g(\mathbf{q}_0, \mathbf{q}_1) \geq 0$,
- **Reflexivity:** $D_g(\mathbf{q}_0, \mathbf{q}_1) = 0 \iff \mathbf{q}_0 = \mathbf{q}_1$,
- **Symmetry:** $D_g(\mathbf{q}_0, \mathbf{q}_1) = D_g(\mathbf{q}_1, \mathbf{q}_0)$,
- **Triangle inequality:** $D_g(\mathbf{q}_0, \mathbf{q}_1) + D_g(\mathbf{q}_1, \mathbf{q}_2) \geq D_g(\mathbf{q}_0, \mathbf{q}_2)$.

Lower bound on $D_g(\mathbf{q}_0, \mathbf{q}_1)$. Let $DISP(\mathbf{q}_0, \mathbf{q}_1)$ be the displacement metric, which is defined as the maximum Euclidean displacement of points on a model at these two configurations. It follows that $DISP(\mathbf{q}_0, \mathbf{q}_1)$ is a lower bound of $D_g(\mathbf{q}_0, \mathbf{q}_1)$:

$$D_g(\mathbf{q}_0, \mathbf{q}_1) \geq DISP(\mathbf{q}_0, \mathbf{q}_1).$$

Upper bound on $D_g(\mathbf{q}_0, \mathbf{q}_1)$. In order to compute an upper bound for a 3D rigid object with translational and rotational DOFs, we first consider computing the D_g of by only varying a single DOF. Then, when we vary all the DOFs simultaneously, the final D_g would be less than or equal to the sum of the D_g 's computed with respect to each DOF [Schwarzer et al. 2005; LaValle 2006].

When Euler angles are used to represent rotation, the upper bound on D_g can be calculated as:

$$D_g(\mathbf{q}_0, \mathbf{q}_1) \leq \Delta(\mathbf{q}_x) + \Delta(\mathbf{q}_y) + \Delta(\mathbf{q}_z) + R_\phi \Delta(\mathbf{q}_\phi) + R_\theta \Delta(\mathbf{q}_\theta) + R_\psi \Delta(\mathbf{q}_\psi), \quad (3)$$

where the Lipschitz constants R_ϕ, R_θ, R_ψ are the maximum Euclidean distances from any point on A to X, Y, Z axes in the local coordinate system, respectively. Δ denotes the difference of each DOF between these two configurations.

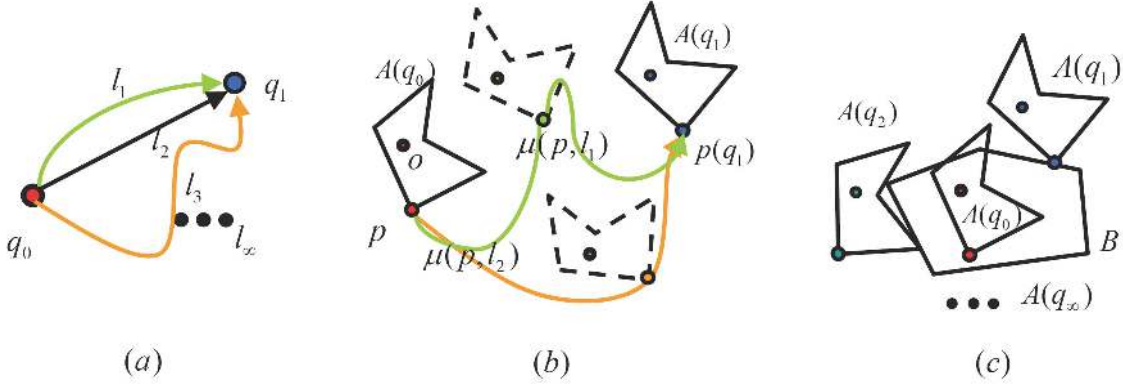


Figure 1: Generalized penetration depth PD^g definition : (a) In the C-space, there are an infinite number of curves, such as l_1, l_2 , that connect two configurations \mathbf{q}_0 and \mathbf{q}_1 . (b) When the configuration of the object A changes along any curve l , any given point on A will trace out a distinctive trajectory in the 3D Euclidean space. This sub-figure shows the trajectories traced by \mathbf{p} when A travels along l_1 and along l_2 , while $\mu(\mathbf{p}, l_1)$ and $\mu(\mathbf{p}, l_2)$ are the arc-lengths of these trajectories, respectively. For each curve l , some point on A corresponds to the longest trajectory length as compared all other points on A . The distance metric $D_g(\mathbf{q}_0, \mathbf{q}_1)$ is defined as the minimum over longest trajectory lengths over all curves connecting \mathbf{q}_0 and \mathbf{q}_1 . (c) PD^g is defined as the minimum of $D_g(\mathbf{q}_0, \mathbf{q})$ over all free configurations, which do not intersect with B (such as $\mathbf{q}_1, \mathbf{q}_2$).

If the rotation vector is used to represent rotation, the upper bound can be calculated by:

$$D_g(\mathbf{q}_0, \mathbf{q}_1) \leq \Delta(\mathbf{q}_x) + \Delta(\mathbf{q}_y) + \Delta(\mathbf{q}_z) + R \sum_{k=1}^3 \Delta r_k, \quad (4)$$

where the constant R is the maximum Euclidean distance from the origin of A to every point on A . In section 6, we use these two upper bound formulae to compute an upper bound on PD^g .

3.3 PD^g Definition

Using D_g metric, we define our generalized PD, PD^g as:

$$PD^g(A, B) = \min(\{D_g(\mathbf{q}_0, \mathbf{q}) \mid \text{interior}(A(\mathbf{q})) \cap B = \emptyset\}) \quad (5)$$

where \mathbf{q}_0 is the initial configuration of A , and \mathbf{q} is in C-space (Fig. 1).

The translational PD^t defined by Eq. (1) is essentially a special case of PD^g . When an object A can only translate, all the points on A traverse the same distance. As a result, the distance metric $D(\mathbf{q}_0, \mathbf{q}_1)$ is equal to the Euclidean distance $\|\mathbf{q}_0 - \mathbf{q}_1\|$. In this case, Eq. (5) can be simplified to Eq. (1). Our generalized PD formulation has a geometric interpretation in C-space. PD^g is realized by some configuration \mathbf{q} on the boundary of free space whose distance (D_g) to the given configuration \mathbf{q}_0 is the minimum.

In terms of handling general non-convex polyhedra, it is difficult to compute PD^g . This is due to the high combinational complexity of C-Space arrangement computation, which can be as high as $O(n^{12})$. However, reducing the problem to only dealing with convex primitives can significantly simplify the problem. In the following sections, we show that if both the input polyhedra are convex, their PD^g is equal to PD^t . Furthermore, if the complement of one of the polyhedra is convex, we reduce PD^g to a variant of a convex containment problem. In case of general non-convex polyhedra, we treat them as a combination of above two cases to compute a lower bound and an upper bound on PD^g .

4 PD^g Computation between Convex Objects

In this section, we consider the problem of computing generalized PD between two convex objects. In this case, we prove that PD^g is

equal to PD^t . As a result, the well known algorithms to compute PD^t between convex polytopes [van den Bergen 2001; Kim et al. 2002b] are directly applicable to PD^g .

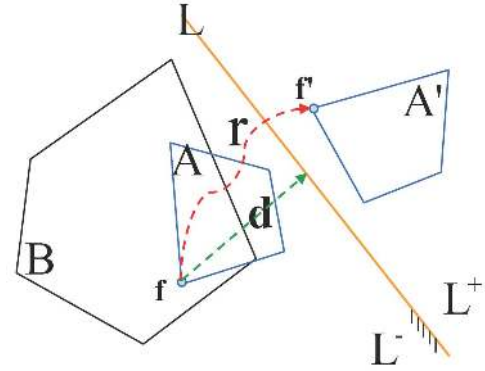


Figure 2: Proof for $PD^t(A, B) = PD^g(A, B)$ for convex objects A and B . Let A' a placement of A which realizes PD^g . L is an arbitrary separating plane between A' and B , which divides the space into two half-spaces L^- and L^+ . For any L , there always exists a point \mathbf{f} on A on L^- side with $\|\mathbf{d}\| \geq PD^t(A, B)$. As a result, we cannot move A towards L^+ side with a traveling distance that is less than $PD^t(A, B)$ even when rotational DOFs are allowed. Therefore, generalized PD is equal to translational PD for convex objects.

Theorem 1 Given two convex objects A and B , we have

$$PD^g(A, B) = PD^t(A, B)$$

Proof Let us assume that A and B intersect, otherwise it is trivial to show that $PD^g = PD^t = 0$.

First of all, we can say that $PD^g \leq PD^t$, as PD^g is realized under more DOFs than PD^t . Next we show that $PD^g < PD^t$ is not possible and therefore, we can conclude $PD^g = PD^t$. We use a proof by contradiction.

Suppose $PD^g < PD^t$. Let us call A' as the placement of A that realizes PD^g , implying that A' is disjoint from B (Fig. 1). Since A' and B are convex, there exists a separating plane L that separates A' and B . Moreover, let L divide the entire space into two half-spaces: L^- , which contains B and L^+ , which contains A' . Let \mathbf{f} be the farthest point on A on L^- side from the separating plane L and \mathbf{d} be

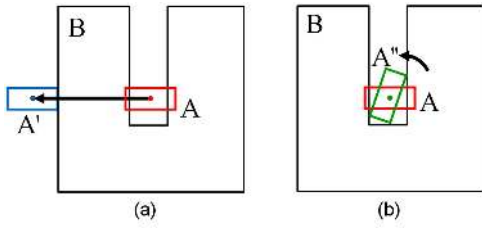


Figure 3: An example of $PD^s < PD^f$ between convex A and non-convex B . The trajectory length that A travels is much shorter when both translation and rotation transformation are allowed (b) than the length when only translation is allowed (a).

the vector from \mathbf{f} to its nearest point on L . As a result, $\|\mathbf{d}\| \geq PD^f$. Otherwise, we could separate A and B by translating A by \mathbf{d} , which would result in a smaller PD^f (i.e. \mathbf{d}) and this contradicts the definition of PD^f in Eq. 1.

Since \mathbf{f} , which is on L^- side, is at least PD^f far away from L , \mathbf{f} must travel at least by PD^f to reach the new position \mathbf{f}' , which can be lying on L or contained in L^+ . However, according to the definition of PD^s in Eq. (5) and the assumption of $PD^s < PD^f$, there must exist a trajectory l connecting \mathbf{f} and \mathbf{f}' , whose arc-length is less than PD^f . This means that \mathbf{f} could be moved to L or within L^+ by less than the amount of PD^f , which is contradictory to the earlier observation that \mathbf{f} must travel at least by PD^f . Therefore, we conclude that L can not be a separating plane between A' and B .

The above deduction shows under the assumption that $PD^s < PD^f$, no separating plane can exist. This contradicts the fact that there must exist a separating plane when convex objects are disjoint. Therefore, $PD^s < PD^f$ is not possible and hence $PD^s = PD^f$. Q.E.D.

Corollary 1 For two convex objects A and B , their generalized PD is commutative; i.e.,

$$PD^s(A, B) = PD^s(B, A).$$

Proof For convex objects, $PD^s(A, B) = PD^f(A, B)$ and $PD^s(B, A) = PD^f(B, A)$. Since PD^f is commutative such that $PD^f(A, B) = PD^f(B, A)$, it follows that $PD^s(A, B) = PD^s(B, A)$. Q.E.D.

Non-Convex objects. Note that, for non-convex objects, $PD^s(A, B)$ is not necessarily equal to $PD^f(A, B)$. Figs. 3 and 4 show such examples. In Fig. 3, $PD^s(A, B) < PD^f(A, B)$, because the trajectory length that any point on A travels is shorter when both translation and rotation transformation are allowed (b) than its corresponding length when only translation is allowed (a). In Fig. 4, an object B , which could be infinitely large with a hole inside, can contain A only when A adjusts its initial orientation. Hence, the $PD^f(A, B) = \infty$ (i.e. the height of B), but $PD^s(A, B)$ is not ∞ (i.e. is much smaller than the height). So, $PD^s(A, B) < PD^f(A, B)$. We can also see that $PD^s(A, B)$ is not necessarily equal to $PD^s(B, A)$ in this example. If B is movable, the D_g metric for B at any two distinctive orientations is always ∞ , because B is unbounded. Therefore, $PD^s(B, A)$ is ∞ in this case, while $PD^s(A, B)$ is not ∞ .

5 PD^s Computation between a Convex Object and a Convex Complement

In this section, we show how to pose the generalized PD computation as a containment problem. Using this formulation, we investigate a special case of generalized PD where a movable object A and the complement of a fixed object B (i.e. \bar{B}) are both convex

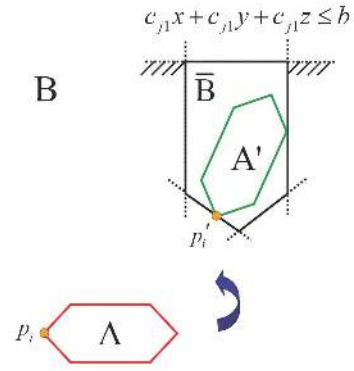


Figure 4: PD^s between the convex object A and the object B whose complement - \bar{B} is convex. In this case, the $PD^s(A, B) \neq PD^f(A, B) = \infty$ and $PD^s(A, B) \neq PD^s(B, A) = \infty$. We compute an upper bound on PD^s by reducing the problem to a variant of the convex containment problem by using linear programming.

(as shown in Fig. 4). Instead of computing an exact solution, we compute an upper bound of PD^s by using a two-level optimization algorithm based on linear programming.

5.1 Relationship between PD^s and Object Containment

The general *object containment problem* can be stated as follows: given two objects P and Q , determine whether Q can contain P by performing translation and rotation transformation on P . The PD^s definition in Eq. (5) is closely related to the object containment problem. That is, testing $interior(A(\mathbf{q})) \cap B = \emptyset$ in Eq.(5) can be reduced to a containment problem: whether \bar{B} can contain A , as shown in Fig. 4. However, there are a few differences between these two problems. The object containment problem finds one instance of a placement of A that can fit inside of \bar{B} , whereas PD^s computation needs to search through all valid containment configurations to find a configuration that minimizes the objective function D_g in Eq. (5).

The standard object containment problem is known to be difficult even for 2D polygonal models. However, if the primitives are convex, computational complexity of containment reduces from $O(m^3n^3 \log(mn))$ to $O(mn^2)$ for polygons with m and n vertices [Chazelle 1983; Avnaim and Boissonnat 1989]. As a result, we consider the case when a movable polyhedron A is convex and the complement of a fixed polyhedra B is convex as well. To compute an upper bound of PD^s for this case, our algorithm performs two levels of optimizations:

1. We compute a configuration \mathbf{q}_1 for A such that the convex container \bar{B} contains $A(\mathbf{q}_1)$. This is performed by minimizing their overlap. The valid containment yields an upper bound of PD^s , which may not be tight.
2. We iteratively compute a configuration, \mathbf{q}_2 , to yield a tighter upper bound of PD^s by setting the upper bound of D_g metric in Eq. (4) as the objective function for optimization.

5.2 Computing a Containment

In this section, we introduce the formulation of the convex containment problem, and extend the 2D optimization-based algorithm described in [Milenkovic 1999; Grinde and Cavalier 1996] to 3D objects, both of which serve as a foundation of finding a locally-optimal containment.

Formulation of 3D Convex Containment. To check whether A fully lies inside \bar{B} can be mathematically formulated as follows. The convex object, \bar{B} with n faces is represented as an intersection of n half-spaces $\mathbf{c}_j \mathbf{x} \leq b_j, j = 1, \dots, n$. A placement of A lies fully inside \bar{B} if and only if every vertex $\mathbf{p}_i (i = 1, \dots, m)$ on A lies inside all the half-spaces, i.e. $\mathbf{c}_j \mathbf{p}_i \leq b_j, i = 1, \dots, m, j = 1, \dots, n$, or:

$$\mathbf{C} \mathbf{p}_i \leq \mathbf{b}, i = 1, \dots, m, . \quad (6)$$

Here \mathbf{c}_j is normalized so that for a given point \mathbf{p} , $|\mathbf{c}_j \cdot \mathbf{p} - b_j|$ is the Euclidean distance from \mathbf{p} to its corresponding face j .

Denote R as the *rotation matrix* when A is rotated around an arbitrary axis with respect to its origin \mathbf{o} . When A is rotated by R , followed by the translation of \mathbf{t} , the new position of \mathbf{p} in A can be calculated as:

$$\mathbf{p}' = R(\mathbf{p} - \mathbf{o}) + \mathbf{o} + \mathbf{t}. \quad (7)$$

Using the above notation, the 3D containment problem now can be stated as finding a solution to the following system:

$$\mathbf{C} \mathbf{p}'_i \leq \mathbf{b}, i = 1, \dots, m. \quad (8)$$

Linearizing 3D Convex Containment Problem. The 3D containment computation is a non-linear problem, as the rotation matrix R is embedded with non-linear terms. These non-linear terms could be linearized by using a *small-angle approximation* [Milenkovic and Schmidl 2001]. When A is rotated by α around an arbitrary axis $\hat{\mathbf{a}}$, its rotation vector \mathbf{r} is equal to $\alpha \hat{\mathbf{a}}$. If the variation of a rotation angle α is small enough, we can get a linearized approximation for Eq. (7) can be obtained:

$$\tilde{\mathbf{p}} \approx \mathbf{p} + \mathbf{r} \times (\mathbf{p} - \mathbf{o}) + \mathbf{t}. \quad (9)$$

By replacing \mathbf{p}' by its approximation $\tilde{\mathbf{p}}$, the non-linear system in Eq. 8 is simplified to a linear one:

$$g_{ij} = \mathbf{c}_j \cdot \mathbf{t} - (\mathbf{c}_j \times (\mathbf{p}_i - \mathbf{o})) \cdot \mathbf{r} + (\mathbf{c}_j \cdot \mathbf{p}_i - b_j) \leq 0, \forall i, j. \quad (10)$$

Here \mathbf{t} and \mathbf{r} are the unknown vectors. g_{ij} , which is called as *containment function*, is defined for each pair of the vertex on A and the face of \bar{B} :

In order to solve the linear system defined in Eq. (10), a slack variable d_{ij} is introduced to represent the distance from $\tilde{\mathbf{p}}_i$, the approximate position of \mathbf{p}_i on A after it is transformed, to the j 'th face on \bar{B} . In this case, the 3D convex containment constraint for A and \bar{B} can be approximated as a linear programming problem (LP1):

$$\begin{aligned} \min Z &= \sum_{i=1}^m \sum_{j=1}^n d_{ij}, \\ \text{subject to } &g_{ij}(\mathbf{t}, \mathbf{r}) - d_{ij} \leq 0 \forall i, j. \end{aligned} \quad (11)$$

If $Z = 0$ for this optimization problem, we end up computing a solution to Eq. (10).

Containment Computation. Given A and \bar{B} , we construct a linear programming problem defined as in Eq. (11) and apply the standard linear programming technique to optimize its objective function Z . We compute the solution, say (\mathbf{t}, \mathbf{r}) , and place A at A' . A new linear programming formulation (like LP1) is constructed for A' and solved iteratively until a local minimum for Z is computed. As the algorithm iterates, the small-angle approximation for the rotation matrix R becomes more accurate. When the objective Z approaches zero, a valid containment of A at configuration \mathbf{q}_1 has been found.

5.3 Computing a Locally-Optimal Containment

The optimization algorithm highlighted above can only find a valid containing placement $A(\mathbf{q}_1)$ for A , which yields an upper bound for PD^g . We perform a second level of optimization to compute an even tighter upper bound for PD^g by using the first level containing placement $A(\mathbf{q}_1)$ as an initial placement for the second level.

Let $\mathbf{q}_0 = (\mathbf{t}_0, \mathbf{r}_0)$ be the initial configuration of A used in the first level optimization. Let $\mathbf{q}_1 = (\mathbf{t}_1, \mathbf{r}_1)$ be the configuration for the containing placement A as a result of the first level optimization. Our goal is to compute $\Delta \mathbf{q} = (\Delta \mathbf{t}, \Delta \mathbf{r})$, such that $\mathbf{q}_2 = (\mathbf{t}_1 + \Delta \mathbf{t}, \mathbf{r}_1 + \Delta \mathbf{r})$ yields another containing placement of A while $D_g(\mathbf{q}_0, \mathbf{q}_2) < D_g(\mathbf{q}_0, \mathbf{q}_1)$.

We perform the second level optimization by setting the upper bound on the D_g metric in Eq. (4) as an optimization objective function. Here we do not choose Eq. (3), because the 3D containment computation uses the notations of rotation vector. By imposing that A needs to be contained by \bar{B} as a hard constraint, we get the system:

$$\begin{aligned} \min Z &= \sum_{k=1}^3 |\Delta t_k + t_{1,k} - t_{0,k}| + R \sum_{k=1}^3 |\Delta r_k + r_{1,k} - r_{0,k}|, \\ \text{subject to } &g_{ij}(\Delta \mathbf{t}, \Delta \mathbf{r}) \leq 0 \forall i, j, \end{aligned} \quad (12)$$

where $t_{0,k}$ and $r_{0,k}$ are, respectively, the k th translational and rotational DOF for an initial configuration of A , and similarly $t_{1,k}$ and $r_{1,k}$ are the k th DOF for a configuration as a result of the first level optimization, and $\Delta t_k, \Delta r_k$ are the variables. Note, now the *containment function* g_{ij} is computed from every vertex of $A(\mathbf{q}_1)$ (instead of $A(\mathbf{q}_0)$) and each face of \bar{B} . Let us further set $u_k = \Delta t_k + t_{1,k} - t_{0,k}$ and $v_k = \Delta r_k + r_{1,k} - r_{0,k}$. In this case, we can rewrite the second level optimization problem in Eq. (12) as:

$$\begin{aligned} \min Z &= \sum_{k=1}^3 |u_k| + R \sum_{k=1}^3 |v_k|, \\ \text{subject to } &g_{ij}^1(\mathbf{u}, \mathbf{v}) \leq 0 \forall i, j, \end{aligned} \quad (13)$$

where g_{ij}^1 is obtained from g_{ij} in Eq. 12 by the change of variables: $\mathbf{u} = \Delta \mathbf{t} + \mathbf{t}_1 - \mathbf{t}_0$ and $\mathbf{v} = \Delta \mathbf{r} + \mathbf{r}_1 - \mathbf{r}_0$.

The objective function in the optimization system (Eq. (13)) contains absolute arithmetic operations. We replace $|u_k|$ with $u_k^+ + u_k^-$ in the objective function, and u_k with $u_k^+ - u_k^-$ in the containment function g_{ij}^1 , where $u_k^+, u_k^- \geq 0$ for $k = 1, 2, 3$. A similar replacement is performed for v_k . After the replacement, finally we formulate this optimization problem as a linear programming problem:

$$\begin{aligned} \min Z &= \sum_k (u_k^+ + u_k^-) + R(v_k^+ + v_k^-), \\ \text{subject to } &g_{ij}^2(u^+, u^-, v^+, v^-) \leq 0 \forall i, j \\ &u_k^+, u_k^-, v_k^+, v_k^- \geq 0, k = 1, 2, 3. \end{aligned} \quad (14)$$

where g_{ij}^2 is obtained from g_{ij}^1 by the change of variables.

By solving Eq. (14), we get $u_k^+, u_k^-, v_k^+, v_k^-$. Using the solution, we can compute $\Delta t_{1,k}^+, \Delta t_{1,k}^-, \Delta r_{1,k}^+, \Delta r_{1,k}^-$, which yields a new configuration \mathbf{q}_2 to replace \mathbf{q}_1 . This process is iterated until the objective Z in Eq. (14) converges to a local minimum. At this stage, since $u_k^+, u_k^-, v_k^+, v_k^-$ are zeroes, our small-angle approximation becomes accurate and A is forced to be disjoint from B . After computing an optimal containing placement \mathbf{q}_2 of A , we compute an upper bound on PD^g using Eq. (3).

6 PD^g Estimation for Non-Convex Objects

In this section, we present our algorithm to efficiently compute a lower bound and an upper bound on PD^g between non-convex objects. Our algorithm is built on the properties of PD^g, presented in Section 4 and Section 5.

6.1 Lower Bound on PD^g

Our algorithm to compute a lower bound on PD^g is based on the fact that PD^g is equal to PD^f for convex polyhedra. As a result, we compute a lower bound of PD^g by first computing the inner-convex covers for each input models. The inner-convex cover refers to a set of convex pieces whose union is a subset of the original model [Milenkovic 1998; Cohen-Or et al. 2002]. Next, we take the maximum value of PD^f's between all pairwise combinations of convex pieces. The overall algorithm proceeds as:

1. As a preprocessing, compute inner-convex covers for A and B i.e., $\cup A_i \subseteq A$ and $\cup B_j \subseteq B$ where A_i, B_j are convex sets, but are not necessarily disjoint from each other.
2. During the run-time query, place A_i at the configuration \mathbf{q} , i.e. compute $A_i(\mathbf{q})$.
3. For each pair of $(A_i(\mathbf{q}), B_j)$ where $i = 1, \dots, M$ and $j = 1, \dots, N$,
 - (a) Perform collision detection to check for overlaps.
 - (b) If the pair overlaps, let $PD_k^g = PD^f((A_i(\mathbf{q}), B_j))$; otherwise $PD_k^g = 0$, where $k = 1, \dots, MN$.
4. Finally, $PD^g = \max(PD_k^g)$ for all k .

6.1.1 Translational Penetration Depth Computation

In our method, the lower bound on generalized PD^g computation is decomposed into a set of PD^f queries among convex primitives. The PD^f between two convex polyhedra can be computed using the algorithms presented in [Cameron 1997; van den Bergen 2001; Kim et al. 2002b]. These methods compute PD^f by calculating the minimum distance from the origin to the surface of the Minkowski sum of the two convex polyhedra.

Since we are computing a lower bound to PD^g, this imposes that the PD^f computation algorithm used by our method should compute an exact value or a lower bound to the PD^f. In particular, the algorithm proposed by Cameron [1997] satisfies this requirement and Gino's algorithm [2001] also provides a tight lower bound.

6.1.2 Acceleration using Bounding Volume Hierarchy

Our lower bound to PD^g computation can be accelerated by employing a standard bounding volume hierarchy. For two disjoint convex pieces, their PD^f corresponds to zero. Typically there are many disjoint pairwise combinations of convex pieces (A_i, B_j) . We detect such disjoint pairs using an oriented bounding box (OBB) [Gottschalk et al. 1996] hierarchy and prune them away.

6.1.3 Analysis

The computational complexity of the lower bound PD^g is determined by the number of convex pieces decomposed from the robot A and the obstacle B , and the geometric complexity of these convex pieces, which is determined by the total number of features of the resulting convex pieces. Let m, n be the number of the convex pieces of A and B , respectively. Let the geometric complexity of the convex pieces of A and B be a and b , respectively. Then, the

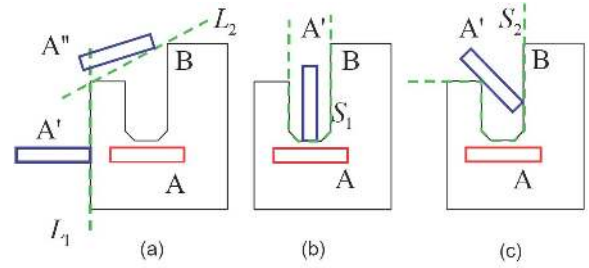


Figure 5: *Separating plane, convex separator and non-convex separator: (a). L_1 and L_2 are separating planes, which separate A' and B , and A'' and B respectively. (b). S_1 is a separator, which is composed by a set of piece-wise linear plane. S_1 separates A' from B . A separator is called convex (i.e. S_1), if it lies on the boundary of its convex hull. (c). A non-convex separator S_2 separates A from B' .*

average numbers of features in each piece of A and B are $\frac{a}{m}$ and $\frac{b}{n}$, respectively. Using computational complexity of translational PD, we can derive that the computational complexity of PD^g for 2D rigid objects is $O(an + bm)$, and for 3D rigid objects is $O(ab)$.

6.2 Upper Bound on PD^g

One simple way to compute an upper bound to PD^g for general non-convex objects is to compute the PD^f between their convex hulls. This corresponds to an upper bound because $PD^g(A, B) \leq PD^g(\text{CH}(A), \text{CH}(B))$, and the latter is equal to $PD^f(\text{CH}(A), \text{CH}(B))$, thanks to Theorem 1. In practice, this upper bound is relative simple to compute. However, this algorithm could be overly conservative for non-convex models, as shown in Figs. 3 and 4.

PD^f(A, B) is also an upper bound on PD^g(A, B). However, this can result in a conservative upper bound in practice. Since the computational complexity of exact computation of PD^f(A, B) for non-convex models can be high, current approaches typically compute an upper bound of PD^f(A, B) [Kim et al. 2002a].

We present an algorithm to compute an upper bound on PD^g for non-convex polyhedra by reducing this problem to a set of containment optimization sub-problems (as defined in Section 5).

6.2.1 Algorithm Overview

Given two disjoint non-convex objects A and B , there is either a single separating plane between the objects (as shown in Fig. 5-(a)) or there is a set of piecewise linear surfaces, which is called a *separator* [Mount 1992]. (Figs. 5 -(b) and -(c)). More precisely, the separator is defined as a simple piece-wise linear surface that divides the space into two half-spaces. The separator can be an open surface or a closed surface. A separator S is convex if and only $S \subset \partial(\text{CH}(S))$, as shown in Fig. 5-(b). Otherwise, the separator is non-convex, as shown in Fig. 5-(c). A single separating plane can be regarded as a special case of a separator. However, we specifically use the term separator to refer to the non-plane separator.

Our upper bound PD^g(A, B) computation algorithm proceeds as follows: during the preprocessing phase, we enumerate all possible separating planes and convex separators by analyzing the convexity of the boundary of B . During the query phase, for each separating plane L (or each convex separator S), we compute an upper bound on D_g distance when A is separated from B with respect to the separating plane L (or separator S) using the technique described in Sec. 5. The minimum over all these upper bounds yields a global upper bound on PD^g. Now we explain how to efficiently enumerate L and S as part of the preprocessing step.

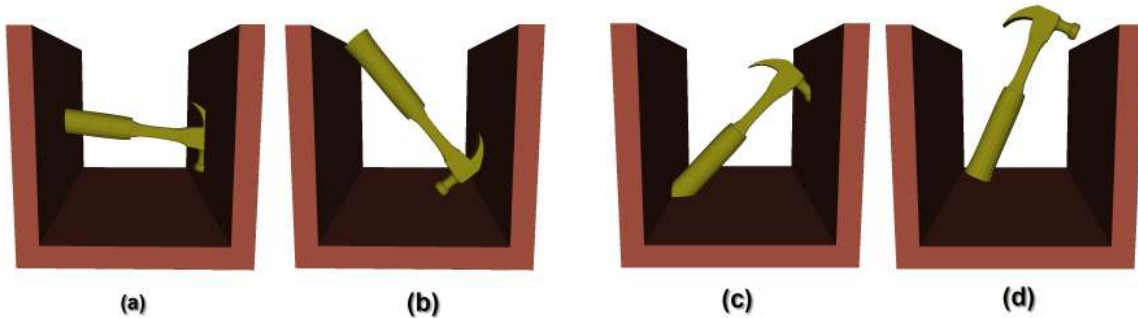


Figure 6: The ‘hammer’ example: (a) When the ‘hammer’ is at time $t=0$, it collides with the ‘notch’. (b) The collision-free placement of the ‘hammer’ for scenario (a). We use our containment optimization algorithm to get this free configuration, which realizes the $UB_1(PD^g)$. (c) The ‘hammer’ at time $t=0.5$. (d) The collision-free placement is computed for scenario to get the $UB_1(PD^g)$

6.2.2 Separating Planes

The set of all possible separating planes is included in the complement of the convex hull of B . According to Theorem 1, $PD^g = PD^f$ for convex objects, and the minimum D_g distance with respect to all these separating planes is $PD^f(CH(A), CH(B))$. This means that the computation of $PD^f(CH(A), CH(B))$ implicitly takes into account all possible separating planes. Therefore, we need not enumerate any separating planes explicitly during the preprocessing phase.

6.2.3 Convex Separators

Any separator S divides the whole space into two half-spaces. One half-space would include the object B . We can regard the other half-space as a container. Placing A inside the container is equivalent to making A and B disjoint with respect to each separator S . Therefore, the computation of the minimum D_g distance for S can be regarded as a 3D convex containment optimization problem. By applying two levels of linear programming optimization algorithm, discussed in Sec. 5, we compute an upper bound of PD^g for each convex separator S . The minimum of all PD^g over all enumerated convex separators yields an upper bound on PD^g .

6.2.4 Convex Separators Enumeration

Enumerating convex separators of B can be performed as a preprocessing. This step can be regarded as computing a convex covering of the complement space of B . Given the fact that we are computing an upper bound of PD^g , the conservativeness of the separator enumeration does not affect the correctness of our algorithm.

We use the surface convex decomposition for the complement space of B [Ehmann and Lin 2001]. We discard the surface with one face from the surface decomposition, since these planes have been processed as separating planes.

Moreover, if the geometry of input A and B is very complex, i.e. high polygon count or a number of features, we compute a simplification of each primitive to compute a coarser model A' , B' . If $A \subseteq A'$ and $B \subseteq B'$, it is easy to prove that $PD^g(A, B) \leq PD^g(A', B')$. Therefore, we can compute the upper bound by applying our algorithm on these simplified models.

6.2.5 Separator Culling

We can cull some of the separators by making use of the currently known upper bound on PD^g during any stage of the algorithm. If the separator is farther away from the object A than the current upper bound, we can discard this separator. We use the PD^f between the two convex hulls of input models as an initial upper bound of PD^g .

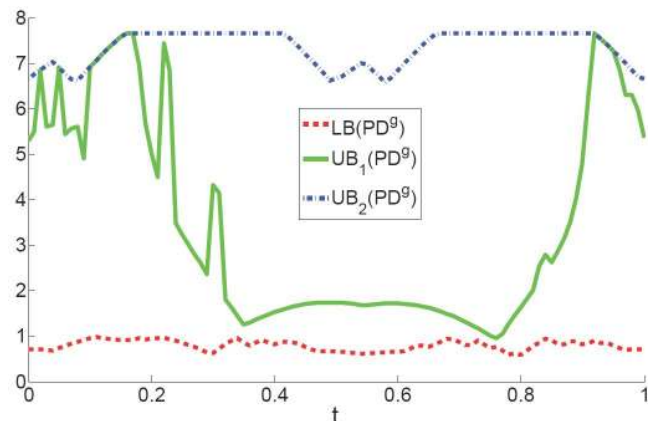


Figure 7: Comparison of lower and upper bounds on PD^g for ‘hammer’ example. The lower and upper bounds on PD^g between the ‘hammer’ and the ‘notch’ models are computed over all interpolated configurations. The dash-dot blue curve $LB(PD^g)$ stands for the lower bound of PD^g by computing pairwise translational PD. The dashed red curve $UB_2(PD^g)$ stands for the upper bound of PD^g computed by the translational PD of their convex hull. The solid green curve $UB_1(PD^g)$ highlights the upper bound of PD^g by using our containment optimization, which always lies between $LB(PD^g)$ and $UB_2(PD^g)$. In this example, $UB_1(PD^g)$ is less than $UB_2(PD^g)$ for most of time t .

7 Implementation and Performance

We have implemented our lower and upper bound computation algorithms for generalized PD computation between non-convex polyhedra. We have tested our algorithms for PD^g on a set of benchmarks, including ‘hammer’ (Fig. 6), ‘hammer in narrow notch’ (Fig. 9), ‘spoon in cup’ (Fig. 8) and ‘pawn’ (Fig. 10) examples. All the timings reported in this section were taken on a 2.8GHz Pentium IV PC with 2 GB of memory.

7.1 Implementation

Lower bound on PD^g . In our implementation, the convex covering is performed as a preprocessing step. Currently, we use the surface decomposition algorithm proposed by [Ehmann and Lin 2001], which can be regarded as a special case of convex covering problem. In order to compute the PD^f between two convex polytopes, we use the implementation available as part of SOLID [van den Bergen 2001]. In order to accelerate this algorithm, we recompute

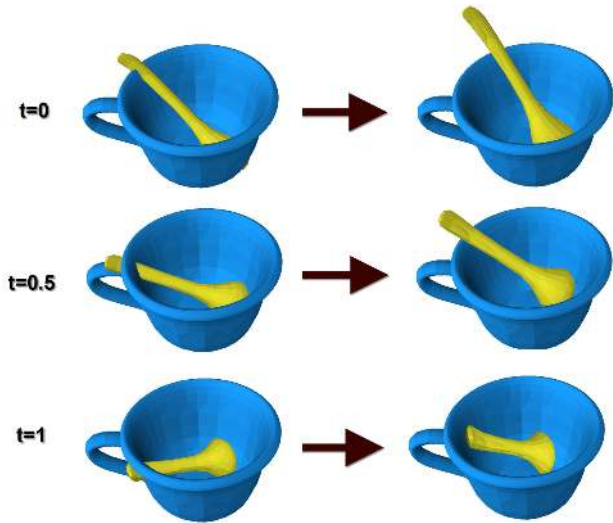


Figure 8: The ‘cup’ example. The left column shows the placements of the ‘spoon’ in the ‘cup’, when $t=0.0$, $t=0.5$, and $t=1.0$, respectively. At all of these placements, the ‘spoon’ collides with the ‘cup’. The right column shows the collision-free configurations which are realized for $UB_1(PD^g)$ at each t .

an OBB hierarchy [Gottschalk et al. 1996] and use the bounding volumes to conservatively cull convex pairs that do not intersect with each other.

Upper bound on PD^g . The preprocessing step of convex separator enumeration can be regarded as convex decomposition of the complement of the input model. In our implementation, we used the surface decomposition algorithm to generate a set of convex surfaces [Ehmann and Lin 2001] and discard the surfaces that have only one face. For each convex separator, we use the containment optimization technique developed in Sec. 5 to compute an upper bound on PD^g . Moreover, we use the *QSopt*¹ package to solve the linear programming problems. In order to accelerate the upper bound computation, we conservatively cull the convex separators that are farther away than the current upper bound on PD^g .

7.2 Performance

We use different benchmarks to test the performance. Our experimental setup is as follows. Each benchmark includes two polyhedral models A and B , where A is movable and B is fixed. The model A is assigned a starting configuration \mathbf{q}_0 and an end configuration \mathbf{q}_1 . We linearly interpolate between these two configurations with n intermediate configurations (i.e. n samples). For each interpolated configuration $\mathbf{q} = (1-t)\mathbf{q}_0 + t\mathbf{q}_1$, $t \in [0, 1]$, we compute various bounds for PD^g between $A(\mathbf{q})$ and B , including:

1. $LB(PD^g)$: The lower bound on PD^g based on pairwise translational PD^f computation.
2. $UB_1(PD^g)$: The upper bound on PD^g computed by containment optimization.
3. $UB_2(PD^g)$. The upper bound on PD^g based on the translational PD^f computation between their convex hull.

¹<http://www2.isye.gatech.edu/~wcook/qsopt/>

In order to get accurate timing profiling, we run our PD algorithms for each configuration with a batch number b . The average time for each bound computation is the total running time on all samples over the product of the number of samples and the batch number b .

‘Hammer’ example. Fig. 6, and Tab. 1 and Fig. 7 show the results and timings for the ‘hammer’ example. In this case, the ‘hammer’ model has 1,692 triangles, which is decomposed into 214 convex pieces. The ‘notch’ model has 28 triangles, which is decomposed into 3 convex pieces and there is a notch (i.e. convex separator) in the center of the ‘notch’ model. Initially (at $t=0$), the ‘hammer’ intersects with the ‘notch’ as shown in Fig. 6(a). Fig. 6(b) shows a collision-free placement of the ‘hammer’, which corresponds to the position after moving by $UB_1(PD^g)$. According to Fig. 7, the value is $UB_1(PD^g) = 4.577083$, which is greater than $LB(PD^g)$ (0.744020) and less than $UB_2(PD^g)$ (6.601070).

For this example, we generate 101 samples for the ‘hammer’ when it is rotated around the Z axis. The rotation motion is linearly interpolated from the configuration $(0, 0, 0)^T$ to $(0, 0, \pi)^T$. Fig. 6(c) shows the placement of the ‘hammer’ at $t = 0.5$. Fig. 6(d) is the corresponding collision-free placement, which realizes the $UB_1(PD^g)$.

We also compare the lower and upper bounds on PD^g over all the configurations. In Fig. 7, the solid green curve highlights the value of $UB_1(PD^g)$ between the ‘hammer’ and the ‘notch’ over all interpolated configurations. The dashed red curve, which corresponds to $UB_1(PD^g)$, always lies between $LB(PD^g)$ and $UB_2(PD^g)$. In this example, $UB_1(PD^g)$ is less than $UB_2(PD^g)$.

The timing for this example is shown in Tab. 1. We run the PD^g algorithm 5 times ($b=5$) for all the configurations ($n=101$). The average timing for $LB(PD^g)$, $UB_1(PD^g)$, and $UB_2(PD^g)$ is 1.901ms, 21.664ms and 0.039ms respectively.

‘Hammer in narrow notch’ example. We perform a similar experiment on ‘Hammer in narrow notch’ example (Fig. 9) to test the robustness of our algorithm. This example is modified from the ‘hammer’ example, where the size of the notch is decreased such that there is only narrow space for the ‘hammer’ to fit inside. Our algorithm can robustly compute the lower and upper bounds on PD for this example. Fig. 11 compare the lower and upper bounds on PD^g over all sampled configurations ($n=101$). The third row of Tab. 1 shows the performance of our algorithm for this example.

‘Spoon in cup’ example. We apply our algorithm on more a complex scenario such as shown in Fig. (8). In this example, the ‘spoon’ model has 336 triangle and is decomposed into 28 convex pieces. The ‘cup’ model has 8,452 triangles. We get 94 convex pieces and 53 convex separators after simplifying the original model to 1,000 triangles.

In Fig. 8, the left column shows the placements of the ‘spoon’ in the ‘cup’, corresponding to $t = 0.0$, $t = 0.5$, and $t = 1.0$, respectively. At all these placements, the ‘spoon’ collides with the ‘cup’. The right column of this figure shows the collision-free configurations that are computed based on $UB_1(PD^g)$ in each case. We also compare our computed lower bound and upper bounds over all the samples ($n=101$), which is shown in Fig. 8. The timing performance for this example is also listed on Tab. 1.

‘Pawn’ example. The last benchmark used to demonstrate the performance of our algorithm is the ‘pawn’ example. As Fig. 10 shows, the large ‘pawn’ is fixed, while the small one is moving. The

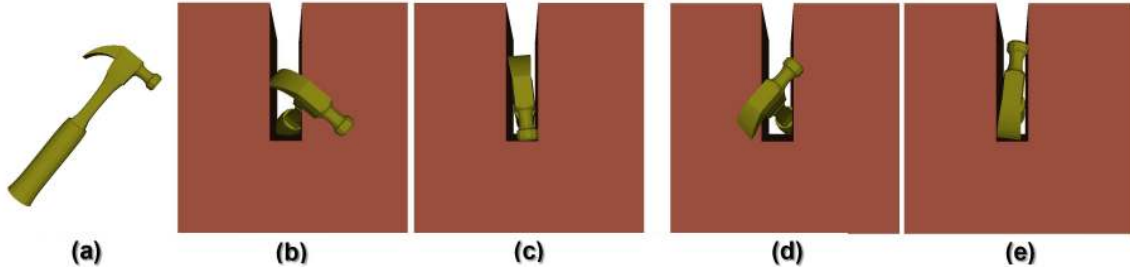


Figure 9: The ‘hammer in narrow notch’ example. This example is modified from the ‘hammer’ example, where the size of the notch is decreased such that there is only narrow space for the ‘hammer’ to fit inside. (b) and (d) shows the placement of the ‘hammer’ at $t=0$ and $t=0.5$. (c) and (e) are their corresponding configurations respectively, which realize the $UB_1(PD^g)$. The computed $UB_1(PD^g)$ is tighter than the $UB_2(PD^g)$ for most of time t .

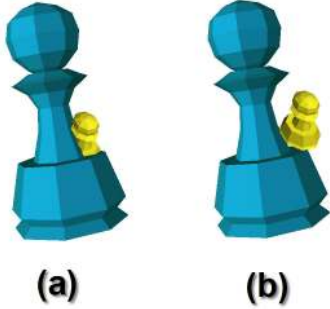


Figure 10: The ‘pawn’ example. The large ‘pawn’ is fixed and the small one is movable. (a) shows the colliding placement of the ‘pawn’ at $t = 0$. (b) shows its corresponding collision-free placement, which is computed based on $UB_1(PD^g)$.

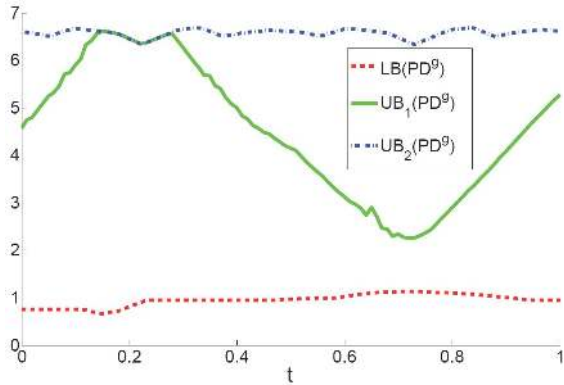


Figure 11: Comparison of lower and upper bounds on PD^g for the ‘hammer in narrow notch’ example.

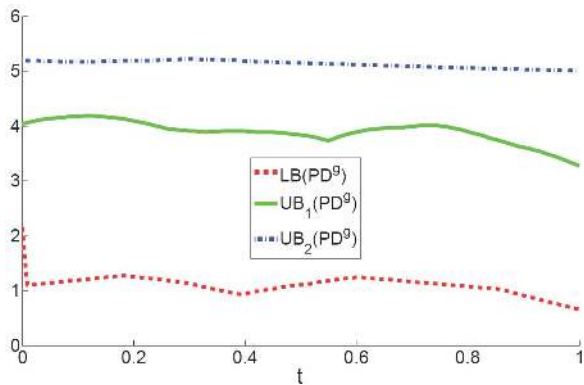


Figure 12: Comparison between lower and different upper bounds on PD^g for ‘cup’ example.

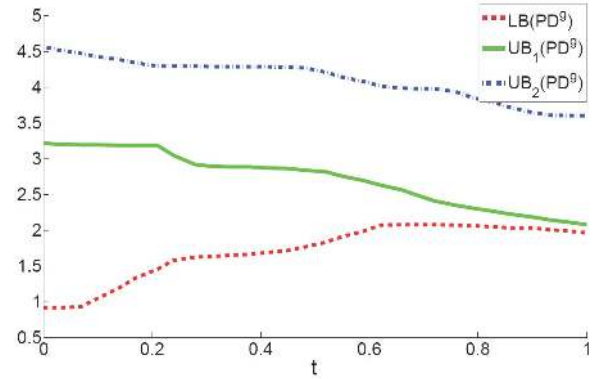


Figure 13: Comparison of lower and upper bounds on PD^g for the ‘pawn’ example.

‘pawn’ model has 304 triangles and is decomposed into 44 convex pieces. The large ‘pawn’ has 43 convex separators. Fig. 10(a) shows the colliding placement of the ‘pawn’ at $t = 0$. Fig. 10(b) shows its corresponding collision-free placement, which is computed based on $UB_1(PD^g)$. 13 compares the lower bound and upper bounds over the sampled configuration ($n=101$). Tab. 1 shows the average time to compute the lower and upper bounds over all configurations.

8 Application to Motion Planning

In this section, we apply our lower bound on PD^g computation algorithm for complete motion planning of planar robots with 3-DOF. The complete motion planning checks for the existence of a collision-free path or reports that no such path exists. It is different from motion planning algorithms based on random sampling, which can not check for path non-existence.

8.1 C-obstacle Query

We mainly use our lower bound on PD^g computation algorithm to perform the *C-obstacle query*. This query for a given C-space is formally defined as checking whether the following predicate P is always true [Zhang et al. 2006b]:

$$P(A, B, Q) : \forall \mathbf{q} \in Q, A(\mathbf{q}) \cap B \neq \emptyset \quad (15)$$

Here, A is a robot, B represents obstacles and Q is a C-space primitive or a cell; $A(\mathbf{q})$ represents the placement of A at the configura-

	Hammer	H2 *	Spoon	Pawn
A	Hammer	Hammer	Spoon	Small
tris #	1,692	1,692	336	304
convex pieces #	215	215	28	44
B	Notch	Notch	Cup	Large
tris #	28	28	8,452	304
convex pieces #	3	3	94	44
separator #	1	1	53	43
sample # (n)	101	101	101	101
batch # (b)	5	5	5	5
t for LB_1 (ms)	1.901	4.300	6.127	4.112
t for UB_1 (ms)	21.664	108.024	1027.014	482.511
t for UB_2 (ms)	0.039	0.053	0.154	0.055

Table 1: This table highlights the benchmarks used to test the performance of our algorithms. The top rows in the table list the model complexity and the bottom rows report the time taken to compute the lower and upper bounds to PD^s on a 2.8GHz Pentium IV PC. ‘H2*’ is the example ‘hammer in narrow notch’.

tion q . Q may be a line segment, a cell or a contact surface that is generated from the boundary features of the robot and the obstacles.

The C-obstacle query is useful for cell decomposition based algorithms for motion planning [Latombe 1991]. These algorithms subdivide the configuration space into cells and need to check whether a cell is fully contained either in the *free space* or in *C-obstacle space*. The free space is the set of all collision-free configurations of the robot. The C-obstacle space is the complement of the free-space. The *C-obstacle query* checks whether a subset of the C-space (i.e. Q) fully lies in the C-obstacle space.

The *C-obstacle query* also arises in sampling based approaches for motion planning, especially complete motion planning. These include the star-shaped roadmap algorithm [Varadhan and Manocha 2005], which is a deterministic sampling algorithm and subdivides the configuration space into a collection of cells in a hierarchical fashion. Given that the time and space complexity of these methods grows quickly with the level of subdivision, it is important to identify cells that lie in C-obstacle space and no further subdivision is executed.

Another benefit of the *C-obstacle query* is to determine non-existence of any collision-free path. The methods in [Zhang et al. 2006a; Varadhan and Manocha 2005] conclude that no path exists between the initial and goal configurations if they are separated by C-obstacle space. These methods can be performed using the *C-obstacle query* to identify these regions which lie in C-obstacle space.

In order to efficiently perform *C-obstacle query* for any cell in C-space, we compute the PD^s by setting its configuration as the center of the cell. Then we compare it with the maximal motion that the robot can undergo when its configuration is confined within a cell [Schwarzer et al. 2005]. If the lower bound of PD^s is larger than the upper bound of the maximal motion, we conclude that the cell (i.e. Q) fully lies in C-obstacle space [Zhang et al. 2006b].

8.2 Experimental Results

We apply our *C-obstacle query* algorithm to improve the performance of a deterministic sampling motion planning algorithm - the star-shaped roadmap method by [Varadhan and Manocha 2005]. To demonstrate the effectiveness of our C-obstacle cell query, we define the *cell culling ratio* as the number of cells in C-obstacle space

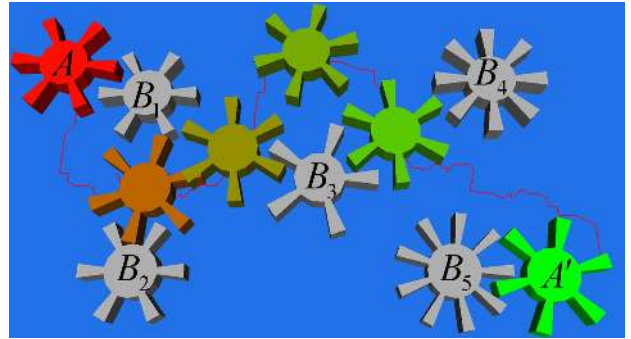


Figure 14: This figure illustrates an application of our C-obstacle query algorithm to speedup a complete motion planner - the star-shaped roadmap algorithm. In this example, the object Gear needs to move from initial configuration A to goal configuration A' by translating and rotating within the shaded rectangular 2D region. We show the robot's intermediate configurations for the found path. Using our C-obstacle query, we can achieve about 2.4 times speed up for the star-shaped roadmap algorithm for this example.

	Gear
Cell Culling Ratio	75.21%
Time Per Cell Culling(ms)	0.12
Time of Original Method(s)	261.4
Time of Accelerated Method(s)	110.4
Speedup	2.4
Time for C-obstacle Cell Query(s)	13.3

Table 2: Performance for C-obstacle Cell Query: For the Gear example, our query can identify about 75.21% C-obstacle cells. The average query time is about 0.12ms. Based on PD^s computation and C-obstacle query, we improve the performance of the star-shaped motion planning algorithm by 2.4 times in this case.

identified by our query algorithm over the total number of cells in C-obstacle space.

Tab. 2 illustrates that our *C-obstacle query* algorithm can achieve 75.21% cell culling ratio in our Gear benchmark. Tab. 2 also shows that the average time for each *C-obstacle query* in the Gear example is about 0.12ms. In this complex 2D scenario, the *C-obstacle query* algorithm improves the performance of the motion planning algorithm by 2.4 times.

9 Limitations

Our PD^s computation algorithm has a few limitations. Given the complexity of exact PD^s computation for non-convex polyhedra, we only compute lower and upper bounds and not the exact answer. Moreover, the convex containment optimization algorithm that linearizes the rotational component can not guarantee a global minimum. The bounds computed by our algorithm also depend on convex covering and separator enumeration of the non-convex polyhedra, performed as part of preprocessing step. As a result, we are unable to provide any tight bounds on the approximation to PD^s computed by our algorithm. However, in most practical cases the extent of penetration is small and we expect that our algorithm would compute a good approximation.

10 Conclusions and Future Work

We have addressed the problem of generalized PD computation between non-convex models, which takes into account translational as well as rotational motion. To the best of our knowledge, this is the first algorithm for general 3D polyhedra models. We present three main results related to PD^g computation. Specifically, we show that for convex models, generalized PD is the same as translational PD. We also present practical algorithms to compute the upper and lower bounds on PD^g for non-convex models.

Our empirical results show that we can efficiently compute the lower and upper bounds of generalized PD for non-convex objects. We also use our algorithm for complete motion planning of polygonal robots with 3-DOF C-space.

Future Work. There are many avenues for future work. On a theoretical side, there are two open questions with respect to generalized penetration depth: how to formulate the distance metric D_g and compute the PD^g for non-convex models in a computational tractable way. It would be useful to derive tight bounds on the approximations (i.e. the lower and upper bounds). Furthermore, we would like to use our algorithm for other applications, including motion planning in 6-DOF C-space, dynamic simulation and tolerance verification.

Acknowledgment

This project was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043 and Intel. Young J. Kim was supported in part by the grant 2004-205-D00168 of KRF, the STAR program of MOST, the Ewha SMBA consortium and the ITRC program. We would also like to thank the anonymous reviewers for their helpful comments.

References

- AGARWAL, P., AMENTA, N., AND SHARIR, M. 1998. Largest placement of one convex polygon inside another. In *Discrete Comput. Geom.*, vol. 19, 95–104.
- AGARWAL, P., GUIBAS, L., HAR-PELED, S., RABINOVITCH, A., AND SHARIR, M. 2000. Penetration depth of two convex polytopes in 3d. *Nordic J. Computing* 7, 227–240.
- AMATO, N., BAYAZIT, O., DALE, L., JONES, C., AND VALLEJO, D. 2000. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *IEEE Transactions on Robotics and Automation*, vol. 16, 442–447.
- AVNAIM, F., AND BOISSONNAT, J. 1989. Polygon placement under translation and rotation. In *ITA*, vol. 23, 5–28.
- CAMERON, S. 1997. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *IEEE International Conference on Robotics and Automation*, 3112–3117.
- CHAZELLE, B. 1983. The polygon containment problem. *Advances in Computing Research* 1, 1–33.
- COHEN-OR, D., LEV-YEHUDI, S., KAROL, A., AND TAL, A. 2002. Inner-cover of non-convex shapes. In *The 4th Israel-Korea Bi-National Conference on Geometric Modeling*.
- DOBKIN, D., HERSHBERGER, J., KIRKPATRICK, D., AND SURI, S. 1993. Computing the intersection-depth of polyhedra. *Algorithmica* 9, 518–533.
- EHMANN, S., AND LIN, M. 2001. Accurate and fast proximity queries between polyhedra using surface decomposition. In *Proc. of Eurographics*.
- GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM Siggraph'96*, 171–180.
- GRINDE, R., AND CAVALIER, T. 1996. Containment of a single polygon using mathematical programming. In *European Journal of Operational Research*, Elsevier Science, vol. 92, 368–386.
- HALPERIN, D. 1997. Arrangements. In *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. CRC Press LLC, Boca Raton, FL, ch. 21, 389–412.
- HALPERIN, D. 2002. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3).
- HALPERIN, D. 2005. Private communication.
- HSU, D., KAVRAKI, L., LATOMBE, J., MOTWANI, R., AND SORKIN, S. 1998. On finding narrow passages with probabilistic roadmap planners. *Proc. of 3rd Workshop on Algorithmic Foundations of Robotics*, 25–32.
- KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2002. Fast penetration depth computation using rasterization hardware and hierarchical refinement. *Proc. of Workshop on Algorithmic Foundations of Robotics*.
- KIM, Y., LIN, M., AND MANOCHA, D. 2002. Deep: Dual-space expansion for estimating penetration depth between convex polytopes. In *Proc. IEEE International Conference on Robotics and Automation*.
- 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.
- KUFFNER, J. 2004. Effective sampling and distance metrics for 3d rigid body path planning. In *IEEE Int'l Conf. on Robotics and Automation*.
- LATOMBE, J. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.
- LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press (also available at <http://msl.cs.uiuc.edu/planning/>). to appear.
- LIN, M., AND MANOCHA, D. 2003. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*.
- MILENKOVIC, V., AND SCHMIDL, H. 2001. Optimization based animation. In *ACM SIGGRAPH 2001*.
- MILENKOVIC, V. 1998. Rotational polygon overlap minimization and compaction. In *Computational Geometry*, vol. 10, 305–318.
- MILENKOVIC, V. 1999. Rotational polygon containment and minimum enclosure using only robust 2d constructions. In *Computational Geometry*, vol. 13, 3–19.
- MIRTICH, B. 2000. Timewarp rigid body simulation. *Proc. of ACM SIGGRAPH*.
- MOUNT, D. 1992. Intersection detection and separators for simple polygons. In *Proc. 8th Annual ACM Sympos. Comput. Geom.*, 303–311.
- RAAB, S. 1999. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *Proc. 15th ACM Symposium on Computational Geometry*, 163–172.
- REDON, S., AND LIN, M. 2005. A fast method for local penetration depth computation. *Journal of Graphical Tools*.
- REQUICHA, A. 1993. Mathematical definition of tolerance specifications. *ASME Manufacturing Review* 6, 4, 269–274.
- SCHWARZER, F., SAHA, M., AND LATOMBE, J. 2005. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Tr. on Robotics* 21, 3 (June), 338–353.
- STEWART, D. E., AND TRINKLE, J. C. 1996. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal of Numerical Methods in Engineering* 39, 2673–2691.
- VAN DEN BERGEN, G. 2001. Proximity queries and penetration depth computation on 3d game objects. *Game Developers Conference*.
- VARADHAN, G., AND MANOCHA, D. 2005. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proc. of Robotics: Science and Systems*.
- ZHANG, L., KIM, Y., AND MANOCHA, D. 2006. A simple path non-existence algorithm for low dof robots. Tech. Rep. 06-006, Department of Computer Science, University of North Carolina at Chapel Hill.
- ZHANG, L., KIM, Y., VARADHAN, G., AND D.MANOCHA. 2006. Fast c-obstacle query computation for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA 2006)*.