Fast Penetration Depth Computation and its Applications

(*http://gamma.cs.unc.edu/DEEP* | *PD* | *PDG*)

Young J. Kim
Dept. of Computer Sci. and Eng.
Ewha Womans Univ., Seoul, Korea
kimy@ewha.ac.kr

Liangjun Zhang Ming C. Lin Dinesh Manocha
Dept. of Computer Sci.
Univ. of North Carolina, Chapel Hill, U.S.A.
{zlj,lin,dm}@cs.unc.edu

Abstract

In this paper, we highlight our past experiences on fast penetration depth computation and its applications in different areas such as physically-based animation, 6DOF haptic rendering and robot motion planning.

Keywords: Penetration Depth, Physically-based Modeling, Haptic Rendering, Robot Motion Planning

1 Introduction

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

Given two inter-penetrating rigid polyhedral models, the penetration measure between them can be defined using different formulations. One of the widely used measures for quantifying the amount of intersection is *penetration depth* (PD). The translational penetration depth (PD^t) between two overlapping models is defined as the minimum translational distance required to separate two intersecting rigid models [Cameron 1997; Cameron and Culley 1986; Dobkin et al. 1993]. However, translational PD is not sufficient in many applications as it does not take into account the rotational motion. We can 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* (PD^g).

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 passage in 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].

In this paper, we highlight our past experiences on developing and implementing different PD algorithms and their applications including the following:

- A highly interactive algorithm to estimate PD^t between convex polytopes in 3D, known as DEEP [Kim et al. 2004].
- A fast algorithm to estimate PD^t between two polyhedral models, utilizing a combination of discretized computations and hierarchical refinement [Kim et al. 2002a].
- A fast algorithm to estimate PD^g between two polyhedral models [Zhang et al. 2006c].
- Applications of PD^t to 6DOF haptic rendering [Kim et al. 2003], physically-based animation [Kim et al. 2002b], and tolerance verification [Kim et al. 2002a].
- Applications of PD^g to robot motion planning [Zhang et al. 2006b; Zhang et al. 2006a].

2 Translational Penetration Depth

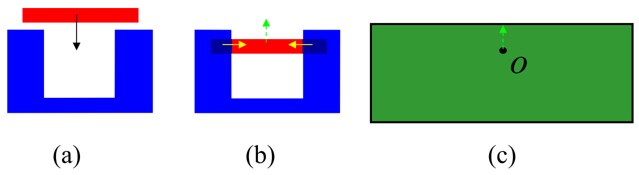

Figure 1: PD Computation. (a) shows the situation before two objects come into contact. (b) shows intersections after the objects are intersected. (c) shows the Minkowski sum of the two objects in (b). The minimum distance from the origin to the surface of the Minkowski sum corresponds to the global PD.

2.1 Definition

The natural extension of Euclidean separation distance to overlapping objects is PD^t. The PD^t of two inter-penetrating objects A and B is defined as the minimum translation distance that one object undergoes to make the interiors of A and B disjoint. Formally, let P and Q be two intersecting polyhedra. Then, the PD of polyhedra P and Q, PD$^t(P,Q)$, is defined as:

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

One of the commonly used metrics for representing and computing PD's is in terms of Minkowski sums of two objects. The Minkowski sums of $A \oplus B$ are defined as a set of pairwise sums of vectors from A and B; i.e.,

$$A \oplus B = \{p+q \mid p \in A, q \in B\} \quad (2)$$

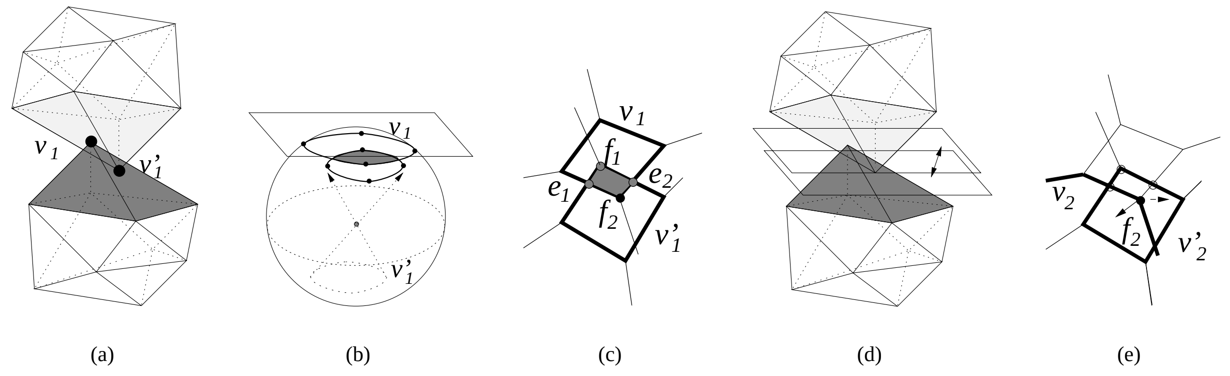

Figure 2: Iterative Optimization: (a) The current (V,V) pair is $v_1v'_1$ and a shaded region represents edges and faces incident to $v_1v'_1$. (b) shows local Gauss maps and their overlay for $v_1v'_1$. (c) shows the result of the overlay after central projection onto a plane. Here, f_1, e_1, f_2 and e_2 comprise vertices (candidate PD features) of the overlay. (d) illustrates how to compute the PD for the candidate PD features in object space. (e) f_2 is chosen as the next PD feature, thus $v_2v'_2$ is determined as the next vertex hub pair.

If we define $-B$ of B as reflecting B with respect to the origin, i.e., $-B = \{-q | q \in B\}$, then the Minkowski sums of $A \oplus -B$, also known as Configuration Space Obstacles (CSO), are defined as

$$A \oplus -B = \{p - q|\ p \in A, q \in B\} \tag{3}$$

Without loss of generality, let us assume that polyhedra A and B are defined with respect to the global origin o. Then, if two polyhedra P and Q intersect, then the origin o is inside of $P \oplus -Q$, and $PD(P,Q)$ corresponds to the minimum distance from o to the surface of the Minkowski sums of $P \oplus -Q$ [Cameron 1997] (see Fig. 1). Thus, the main ingredient of our PD^t algorithms explained in the following boils down to how to compute the Minkowski sums (or CSO) of two intersecting objects in an implicit or approximate manner.

2.2 Algorithm for Convex Polytopes

Our algorithm uses a number of techniques to initialize some features on the surface of the CSO, which are presumably very close to the features that realize the optimum PD. Then, the algorithm incrementally marches towards a “locally optimal” solution by walking on the surface of the CSO. We define the locally optimal PD using the features on the CSO as follows. Let f be a feature on the CSO that corresponds to the locally optimal PD. Then, the distance from the origin to f is always smaller than the distance from the origin to any neighboring feature of f on the CSO.

We implicitly compute the surface of the CSO by constructing a local Gauss map and performing a local walk on the polytopes. Our algorithm performs incremental computations and exploits spatial and temporal coherence between successive frames. Our approach for locally computing the Gauss map is based on an earlier algorithm for width computation between convex polytopes [Kim et al. 2004].

In our incremental PD computation algorithm, we do not compute the entire Gauss map for each polytope or their entire Minkowski sums. Rather we compute them in a lazy and incremental manner, based on local optimization. Starting from some feature on the surface of the Minkowski sums, the algorithm finds the direction in which it can decrease the PD value and proceeds towards that direction by extending the surface of the Minkowski sums.

At each iteration of the algorithm a vertex is chosen from each polytope to form a pair. We label it as a *vertex hub pair* and use it as a hub of the expansion of the local Minkowski sums. The vertex hub pair is chosen in such a way that there exists a plane supporting each polytope, and it is incident on each vertex. It turns out that the vertex hub pair corresponds to two intersected convex regions on the Gauss map, which later become intersecting convex polygons on the plane after *central projection*. The intersection of convex polygons correspond to the VF or EE antipodal pairs from which one can reconstruct the local surface of the Minkowski sums around the vertex hub pair. Given these pairs, we choose the one that corresponds to the shortest distance from the origin of the Minkowski sums to their surface. If this pair decreases the estimated PD value, we update the current vertex hub pair to an appropriate one which is adjacent to the chosen antipodal pair. We iterate this procedure until we can not decrease the current PD value and converge to a local minima; see Fig. 2. This algorithm has been implemented and can be downloaded from *http://gamma.cs.unc.edu/DEEP*. In our experiments, the algorithm is able to estimate the penetration depth in about a milli-second on an 1 GHz Pentium PC. Moreover, its performance is almost independent of model complexity in environments with high coherence between successive instances.

2.3 Algorithm for General Polyhedra

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

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

After the second step, there can be $O(n^2)$ pairwise Minkowski sums and their union can have $O(n^6)$ complexity [Aronov et al. 1997]. This approach provides an algorithmic framework to compute the Minkowski sum.

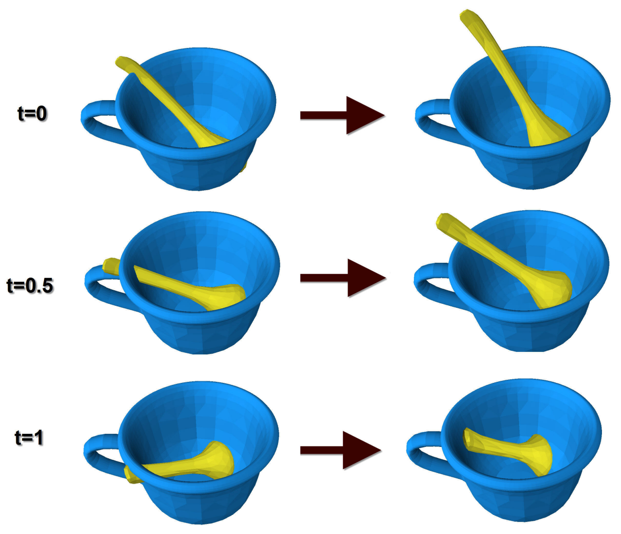

Figure 3: PD^g example. The left column shows the placements of the 'spoon' in the 'cup', when t=0.0, t=0.5, and t=1.0, respectively. At all of these placements, the 'spoon' collides with the 'cup'. The right column of this figure shows the collision-free configurations that are computed based on $UB_1(\mathrm{PD}^g)$ for each t.

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

We improve the performance of the algorithm using a number of acceleration techniques. These include hierarchical representation based on convex bounding volumes, use of model simplification algorithms, and geometry culling approaches applied to both Minkowski sum computation and hardware assisted ray-shooting [Kim et al. 2002a]. This algorithm has been implemented and tested on different benchmarks. Depending on the combinatorial complexity of polyhedra and their relative configuration, its performance varies from a fraction of a second to a few seconds on a 1.6GHz PC with an NVIDIA GeForce 3 graphics card.

3 Generalized Penetration Depth

In this section, we define a generalized PD, PD^g by taking into account translational as well as rotational motion to separate the overlapping objects. We present fast algorithms to estimate a lower bound and an upper bound for PD^g.

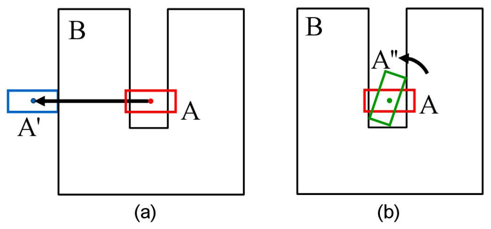

Figure 4: An example of $\mathrm{PD}^g < \mathrm{PD}^t$ between convex A and non-convex B. The trajectory length that A travels is much shorter when both translation and rotation transformation are allowed (b) than the length when only translation is allowed (a).

3.1 Definition

In order to define 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 placed at two different configurations.

Let l be a curve in C-space, which connects two configurations $\mathbf{q}_0$ and $\mathbf{q}_1$ 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 We denote the arc-length of this trajectory as *trajectory length* $\mu(\mathbf{p}, l)$. 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.

$$D_g(\mathbf{q}_0, \mathbf{q}_1) = min(\{max(\{\mu(\mathbf{p}, l) | \mathbf{p} \in A\}) | l \in L\}), \tag{4}$$

where L is a set of all the curves connecting $\mathbf{q_0}$ and $\mathbf{q_1}$. Using D_g metric, we define our generalized PD, PD^g as:

$$\mathrm{PD}^g(A, B) = min(\{D_g(\mathbf{q}_0, \mathbf{q}) | interior(A(\mathbf{q})) \cap B = \emptyset\}) \tag{5}$$

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

The translational PD 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).

3.2 Algorithms

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 [Zhang et al. 2006c], we prove the following theorem:

THEOREM 1 *Given two convex objects A and B, we have*

$$\mathrm{PD}^g(A, B) = \mathrm{PD}^t(A, B)$$

Furthermore, if the complement of one of the polyhedra is convex, we reduce PD^g to a variant of a convex containment problem [Zhang et al. 2006c]. 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.

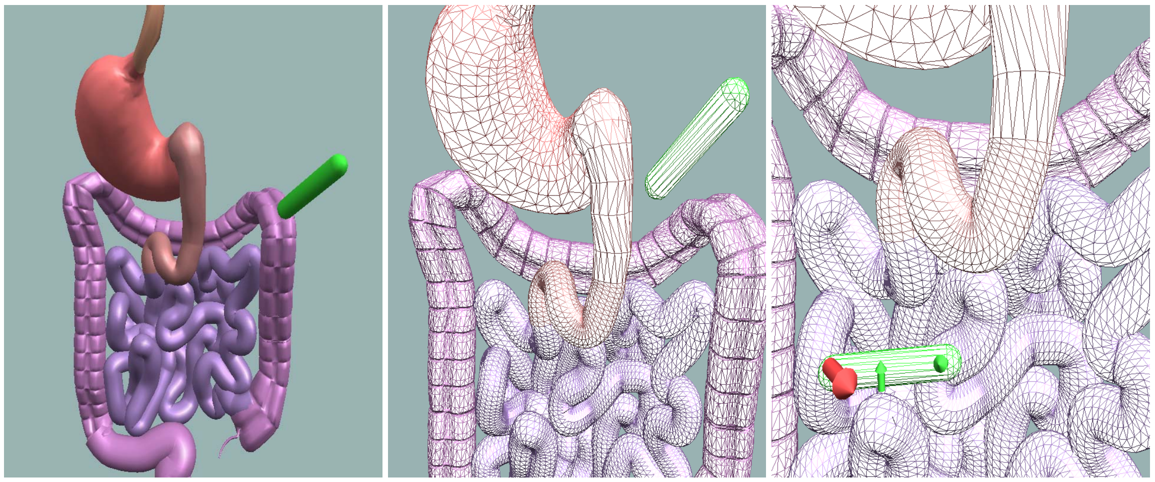

Figure 5: Application of PD^t to Virtual Exploration of a Digestive System Model (25K triangles). Figures show the forces computed at clustered contacts, for both disjoint (D, green arrows) and penetrating (P, red arrow) situations.

3.2.1 Lower Bound on PD^g

Our algorithm to compute a lower bound on PD^g is based on Thm. 1. We compute a lower bound of PD^g by first decomposing each input models into convex pieces [Milenkovic 1998]. Next, we take the maximum value of PD$_i^t$'s between all pairwise combinations of convex pieces. The overall algorithm is stated in Alg. 1.

Algorithm 1 Lower bound of PD^g computation
Input: The robot A, the obstacle B and the configuration $\mathbf{q}$
Output: The lower bound of PD^g between $A(\mathbf{q})$ and B.

```
1:  {During preprocessing}
2:  Decompose $A$ and $B$ into $m$ and $n$ convex pieces; i.e., $A = \cup A_i$
    and $B = \cup B_j$.
3:  {During run-time query}
4:  for each pair of $(A_i(\mathbf{q}), B_j)$ do
5:      $k = (i-1)n + j$
6:      if $A_i(\mathbf{q})$ collides with $B_j$ then
7:          $PD_k^g = PD^t((A_i(\mathbf{q}), B_j)$
8:      else
9:          $PD_k^g = 0$
10:     end if
11: end for
12: return max($PD_k^g$) for all $k$.
```

Our lower bound to PD^g computation can be accelerated by employing a standard bounding volume hierarchy. For two disjoint convex pieces, their PD^t 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.

3.2.2 Upper Bound on PD^g

One simple way to compute an upper bound to PD^g for general non-convex polyhedra is to compute the PD^t between their convex hulls, thanks to Thm. 1. Though this upper bound is relatively simple to compute, it could be overly conservative for non-convex models, as shown in Fig. 4. PD$^t(A,B)$ can be also an upper bound on PD$^g(A,B)$. However, this can result in an overly conservative upper bound too. Since the computational complexity of exact computation of PD$^t(A,B)$ for non-convex models can be high, current approaches typically compute an upper bound of PD$^t(A,B)$ [Kim et al. 2002a].

In order to compute a tighter upper bound on PD^g for non-convex polyhedra, we first investigate a special case that the movable object A and the complement of a fixed object B (i.e $\bar{B}$) are both convex. For this special case, we can compute an upper bound on PD^g using a two-level containment optimization algorithm based on linear programming [Zhang et al. 2006c].

Now for general non-convex polyhedra, our upper bound PD$^g(A,B)$ algorithm can be stated as follows. During the preprocessing phase, we enumerate all possible convex separators by analyzing the convexity of the boundary of B. Here the separator is defined as a simple piece-wise linear surface that divides the space into two half-spaces. A separator S is convex if and only $S \subset \partial(CH(S))$. During the query phase, for each convex separator S, we compute an upper bound on D_g distance when A is separated from B with respect to S. In fact, this computation is equivalent to an upper bound PD^g computation for the special case discussed above, which can be efficiently optimized by using linear programming. The minimum over these upper bounds for all convex separator together with the PD^t yields a tighter upper bound on PD^g.

4 Applications

Now we demonstrate various applications of PD^t and PD^g algorithms.

4.1 6DOF Haptic Rendering

A novel six-degree-of-freedom haptic rendering algorithm has been proposed based on incremental and localized contact computations. It uses an incremental approach for contact and force computations and takes advantage of spatial and temporal coherence between successive frames. As part of a preprocess, we decompose the surface of each polyhedron into convex pieces and construct bounding volume hierarchies to perform fast proximity queries. Once the objects have intersected, we compute PD^t in the neighborhood of the contact between each pair of decomposed convex pieces using the DEEP algorithm. Moreover, we cluster different contacts based on their spatial proximity to speed up the force computation. We have implemented this algorithm and applied it to complex contact scenarios consisting of multiple contacts, as shown in Fig. 5.

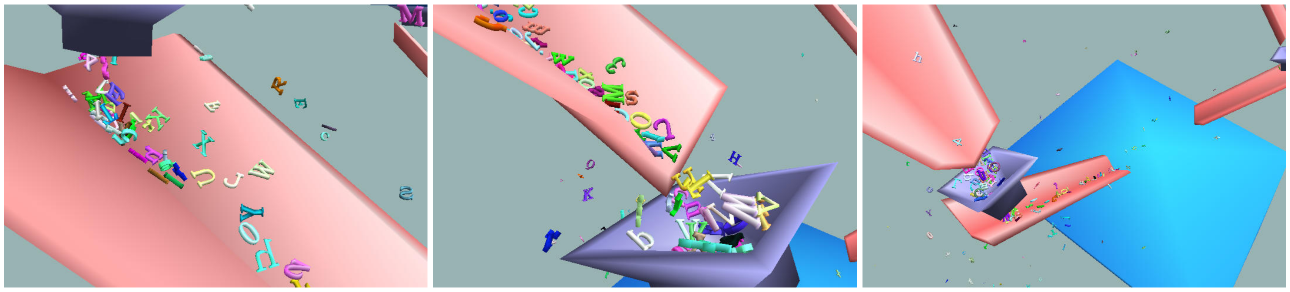

Figure 6: Application of PD^t to Rigid-Body Dynamic Simulation. Our algorithm is used to perform smarter time stepping in a dynamic simulation. A sequence of snapshots are taken from a rigid-body simulation of 200 models of letters and numerical digits falling onto a structure consisting of multiple ramps and funnels.

4.2 Physically-based Animation

PD computation is often needed for dynamic simulation of rigid body systems. In the physical world, objects do not occupy the same spatial extent. However, this is often unavoidable in numerical simulations. For applications involving articulated joints, stacking objects and parts assembly, bodies are nearly in contact or actually touching each other all the time. Our PD^t computation algorithm for non-convex polyhedra provides a consistent and accurate measure of PD.

For example, in penalty-based methods, the forces between rigid bodies are proportional to the amount of inter-penetration. We use PD^t to quantify such an amount of inter-penetration. In constrained-based dynamics, we utilize the knowledge about PD^t features and direction and use a cubic interpolation scheme to estimate the time of collision; see Fig. 6.

4.3 Robot Motion Planning

PD^g computation can be utilized for complete motion planning of polygonal robots undergoing translational and rotational motion. 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.

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]:

$$P(A,B,Q): \ \forall \mathbf{q} \in Q, \ A(\mathbf{q}) \cap B \neq \emptyset \tag{6}$$

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 configuration $\mathbf{q}$. 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.

In order to efficiently perform *C-obstacle query* for any cell in C-space, we compute the PD^g 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. 2002]. If the lower bound of PD^g 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].

The *C-obstacle query* is useful for cell decomposition based algorithms for motion planning [Latombe 1991] and sampling based approaches such star-shaped roadmaps. Fig. 7 illustrates *PD^g* based *C-obstacle query* algorithm is utilized to speed up a complete motion planner - star-shaped roadmaps.

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

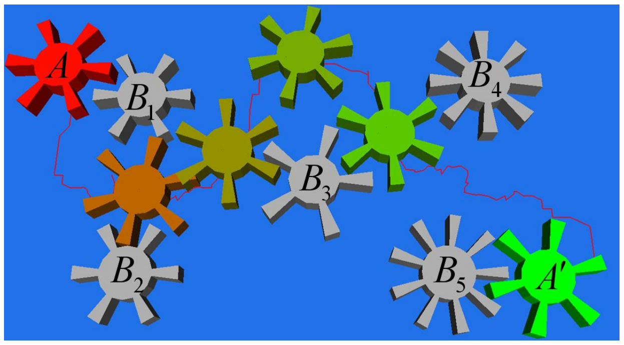

Figure 7: This figure illustrates an application of our *PD^g* based C-obstacle query algorithm to speedup a complete motion planner - the star-shaped roadmap algorithm. In this example, the object *Gear* needs to move from initial configuration A to goal configuration A' by translating and rotating 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.

between the initial and goal configurations if they are separated by C-obstacle space, as shown in Fig 8. These methods can be performed using the *C-obstacle query* to identify these regions which lie in C-obstacle space.

5 Conclusion and Future Work

We have presented efficient algorithms for computing PD^t for convex polytopes, PD^t for general polyhedra and PD^g for general polyhedra. We also have demonstrated their applications to 6DOF haptic rendering, physically-based animation and robot motion planning. For future work, there are a few directions that we will like to further pursue. The main bottle neck in PD^t for general polyhedral lies in the closest point query based on rasterization hardware. This process can be substantially improved by exploiting programmability of modern graphics hardware. The current method of estimating PD^g for general polyhedra is rather loose. It will be quite useful to find a tighter bound of PD^g and apply the PD^g algorithm to other applications such as physically-based animation.

Acknowledgements This project is 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

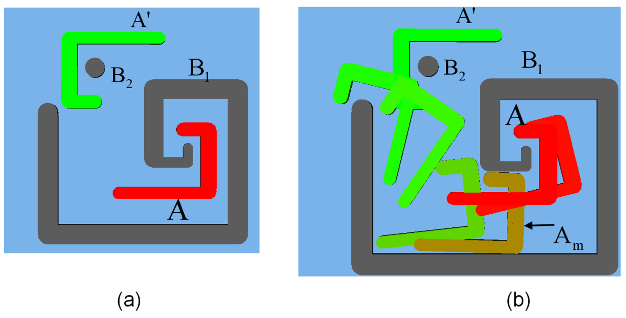

Figure 8: '2D puzzle' example. (a) Our PD^g based C-obstacle query algorithm can be used to check the path non-existence for this planning problem, which needs to move A to A' without colliding with the obstacles B_1 and B_2. (b) is a modified version of (a) with the obstacle B_1 enlarged. A collision-free path through a narrow passage among the obstacles is found. The intermediate configurations of the robot along the collision free path, such as A_m, are also displayed.

J. Kim is sponsored in part by the grant KRF-2004-205-D00168 of the KRF funded by the Korean government, the ITRC program funded by the MIC, and the MOST STAR program.

References

ARONOV, B., SHARIR, M., AND TAGANSKY, B. 1997. The union of convex polyhedra in three dimensions. *SIAM J. Comput. 26*, 1670–1688.

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

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

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

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

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. 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.

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

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

MILENKOVIC, V. J. 1998. Rotational polygon overlap minimization and compaction. *Computational Geometry: Theory and Applications 10*, 305–318.

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

REQUICHA, A. 1993. Mathematical definition of tolerance specifications. *ASME Manufacturing Review 6*, 4, 269–274.

SCHWARZER, F., SAHA, M., AND LATOMBE, J.-C. 2002. Exact collision checking of robot paths. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*.

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.

VARADHAN, G., AND MANOCHA, D. 2005. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proceedings of Robotics: Science and Systems*.

ZHANG, L., KIM, Y. J., AND MANOCHA, D. 2006. A simple path non-existence algorithm for low dof robots. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR 2006)*.

ZHANG, L., KIM, Y. J., VARADHAN, G., AND MANOCHA, D.. 2006. Fast c-obstacle query computation for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA 2006)*.

ZHANG, L., KIM, Y. J., VARADHAN, G., AND MANOCHA, D.. 2006. Generalized penetration depth computation. In *ACM Solid and Physical Modeling Symposium (SPM06)*.