Efficient Penetration Depth Computation between Rigid Models using Contact Space Propagation Sampling

Liang He[1] and Jia Pan[2] and Danwei Li[1] and Dinesh Manocha[1]

Abstract— **We present a novel method to compute the approximate global penetration depth (PD) between two non-convex geometric models. Our approach consists of two phases: offline precomputation and run-time queries. In the first phase, our formulation uses a novel sampling algorithm to precompute an approximation of the high-dimensional contact space between the pair of models. As compared with prior random sampling algorithms for contact space approximation, our propagation sampling considerably speeds up the precomputation and yields a high quality approximation. At run-time, we perform a nearest-neighbor query and local projection to efficiently compute the translational or generalized PD. We demonstrate the performance of our approach on complex 3D benchmarks with tens or hundreds of thousands of triangles, and we observe significant improvement over previous methods in terms of accuracy, with a modest improvement in the run-time performance.**

I. Introduction

Accurate and efficient computation of inter-penetration is important in many areas, including computer graphics, haptics, and robotics. A common metric that is used to measure the extent of inter-penetration between two intersecting objects is *penetration depth* (PD), which is defined as the minimum amount of movement or transformation required to separate two in-collision objects. The resulting motion may correspond to translational alone (translational PD) or to both translational and rotational motion PD (generalized PD). PD computation is frequently used for many applications, such as physically-based simulation [1], sample-based motion planning [2], haptics [3], [4], and contact manipulation [5].

Computing the exact PD in 3D is a challenging task because of the $\mathcal{O}(m^3n^3)$ time complexity involved in translational PD and the $\mathcal{O}(m^6n^6)$ worst case time complexity for generalized PD, where m and n are the number of triangles in two non-convex input models [2]. Given the high combinatorial complexity of exact PD computation, many approximate algorithms have been proposed. Some of the simplest algorithms compute the intersecting features of these two models and use them to compute local PD that is based on a measure of separating those overlapping features. In fact, current game engines such Box2D [6] and Bullet [7] use local PD computations for collision response. However, the accuracy of local PD algorithms depends on relative configuration of two objects [8], [9]. Other techniques are based on computing an approximation of the configuration space boundary [10], [11], [12], but the accuracy of these techniques can vary for different configurations of two objects and it is hard to derive tight bounds. There are no good reliable algorithms for global PD computation between arbitrary non-convex 3D shapes.

Main Results: In this paper, we present a novel algorithm to approximate global PDs between rigid objects based on efficient sampling in the contact space. Our approach can compute both translational and generalized PD with high accuracy for non-convex models. We first precompute an approximation of the contact space of two overlapping objects by generating samples in the contact space. We generate our initial samples using random sampling and use a novel propagation algorithm to generate additional samples via local search. The use of propagation sampling considerably speeds up precomputation and results in a high quality contact space approximation. At run-time, our algorithm performs a nearest-neighbor query to compute the PD. We also analyze the properties of our sampling scheme and highlight its benefits. Compared with prior PD algorithms, our approach offers the following benefits:

- The overall algorithm is general and directly applicable to complex non-convex and non-manifold models.It can compute translational and generalized PDs.
- The use of propagation sampling can considerably accelerate the precomputation and provides a high quality approximation of the contact space.
- The run-time query is very fast (a few milliseconds) and can be used for interactive applications.
- The overall algorithm is more accurate as compared to prior local and global PD computation algorithms.

We highlight the performance of our algorithm on different models, which contain tens or hundreds of thousands of triangles with sharp features. We also highlight the considerable improvements in the accuracy of the run-time query compared with recent algorithms based on active learning [10] and local optimization [13], [4]. In particular, our approach can considerably reduce the error in PD computations over these methods.

The remainder of this paper is organized as follows. In Section II, we survey the literature related to the configuration space and PD computation. We introduce our notation and give an overview of the algorithm in Section III. We present our contact space sampling algorithm in Section IV, and we discuss and analyze its performance on many complex benchmarks in Section V and Section VI.

*This research is supported in part by ARO Contract W911NF-14-1-0437 and NSF award 1305286.

[1]Liang He, Danwei Li and Dinesh Manocha are with the Department of Computer Science, the University of North Carolina at Chapel Hill

[2]Jia Pan is with the Department of Mechanical Engineering and Biomedical Engineering, the City University of Hong Kong.

II. Related Work

A. Configuration Space Computation

There is extensive work on configuration space computations in robotics, geometric computing, and related areas. In the most general cases, configuration space computations can be reduced to computing the arrangement of contact surfaces [14]. However, these approaches are susceptible to robustness issues. Moreover, the worst-case complexity of the arrangement computation can be as high as $\mathcal{O}(n^k)$, where n is the number of contact surfaces in the arrangement and k is the dimension of the configuration space [15]. Some techniques for approximating the configuration space in lower dimensions are based on generating a discrete number of slices [16]. When the object movement is limited to translational motion, the resulting configuration space corresponds to the Minkowski sum of two objects [17], [18].

A substantial amount of work in motion planning involves approximating the configuration space with sampling techniques. These include various randomized algorithms that compute roadmaps for collision-free path planning. Some of these approaches, such as [19], [20], [21], [22], also consider the problem of sampling in a constrained configuration space or a manifold, which is similar to the contact space sampling discussed in this paper. However, these methods are designed for collision-free motion planning, and hence only require the generated samples to capture the connectivity of the free part of the constrained configuration space.

B. PD Computation

Given two convex polytopes, we can compute the exact translational PD using Minkowski sum computation [23], [24], [25]. For non-convex polyhedral models, the PD can be computed using a combination of convex decomposition, pairwise Minkowski sums, and exact union computation [11]. Different techniques have been proposed to approximate the boundary of the Minkowski sum [11], [14], but they are limited to offline and non-interactive applications. Most practical algorithms for translational PD are based on local computations. These local algorithms only consider the intersecting or overlapping of features such as the vertices, edges, and faces. Based on pairwise intersections, they tend to estimate a motion that would separate these intersecting features [26], [9], [12], [27], [28], [29]. Other techniques estimate the local intersection volume and its derivative to perform volume-based repulsion [30]. Local translational PD computation can also be estimated using distance fields [8]. Point-based Minkowski sum approximation [31] has been used approximate the translational PD. The exact computation of generalized PD can be formulated in terms of computing the arrangement of contact surfaces [2]. However, no practical algorithms are known for exact computation due to its high combinatorial complexity. Most practical algorithms are based on local optimization techniques [32], [33], [4], [13]. However, due to the high time and storage complexity, most generalized PD algorithms are based on local optimization-based techniques [32], [33], [4], [13]. [10] recently proposed a learning-based approximate penetration depth computation algorithm that reduces the contact space problem to robust classification by finding a separating surface between in-collision and collision-free samples in the configuration space. However, this algorithm cannot provide high quality approximation of the contact space of objects with sharp features, because it represents the contact space using the SVMs (support vector machines). Recently, Kim et al. [34] present a hybrid PD computation algorithm that combines this active learning approach with local optimization based methods to improve its accuracy.

III. Background and Overview

In this section, we introduce our notation and give an overview.

A. Contact Space

We denote the configuration space for a pair of triangular meshes A and B as $\mathcal{C}$-space. Each configuration or point in the configuration space represents the relative transform (i.e., position and orientation) of A with respect to B. In the rest of this paper, we assume that A is movable and B is fixed. The configuration space is composed of two parts: collision-free space represented as $\mathcal{C}_{\text{free}} = \text{cl}(\{\mathbf{q} : A(\mathbf{q}) \cap B = \emptyset\})$, and in-collision or obstacle space represented as $\mathcal{C}_{\text{obs}} = \text{int}(\{\mathbf{q} : A(\mathbf{q}) \cap B \neq \emptyset\})$, where $A(\mathbf{q})$ corresponds to A located at the configuration $\mathbf{q}$, and $\text{cl}(\cdot)$ and $\text{int}(\cdot)$ correspond to set closure and interior operations, respectively.

The boundary of $\mathcal{C}_{\text{free}}$ is called the *contact space* and is denoted as $\mathcal{C}_{\text{cont}} = \partial\mathcal{C}_{\text{free}}$. The contact space corresponds to the configurations where A and B just touch each other without any penetration. Moreover, a contact configuration is classified as a collision-free configuration in our formulation.

B. PD Formulation

The global penetration depth corresponds to the minimum motion or transformation required to separate two intersecting objects A and B [24], [25]:

$$\text{PD}(A(\mathbf{q}_0), B) = \min_{\mathbf{q} \in \mathcal{C}_{\text{cont}}} \text{dist}(\mathbf{q}_0, \mathbf{q}), \tag{1}$$

where $\mathbf{q}_0$ corresponds to an in-collision configuration and $\mathbf{q}$ is a configuration that belongs to the contact space $\mathcal{C}_{\text{cont}}$. We use the notation $\text{dist}(\cdot,\cdot)$ to represent a distance metric between two configurations. This includes the Euclidean metric for translational PD, and many different formulations can be used for generalized PD computation. We denote $\mathbf{q}_c$ as the contact configuration where $\text{PD}(A, B)$ achieves its minimal value: $\mathbf{q}_c = \text{argmin}_{\mathbf{q} \in \mathcal{C}_{\text{cont}}} \text{dist}(\mathbf{q}_0, \mathbf{q})$.

Different formulations of PD can be defined by appropriate $\text{dist}(\cdot,\cdot)$ metrics. The metric for the translational motion (PD_t) is simple and is the standard Euclidean distance metric between vectors corresponding to the configurations. The metric for the general motion (PD_g) can be defined using different formulations, including the weighted Euclidean distance [3], object norm [35], [4], [13], and a displacement distance metric [33]. In this paper, we use the object norm [35], [4], [13] as the PD_g metric, which can

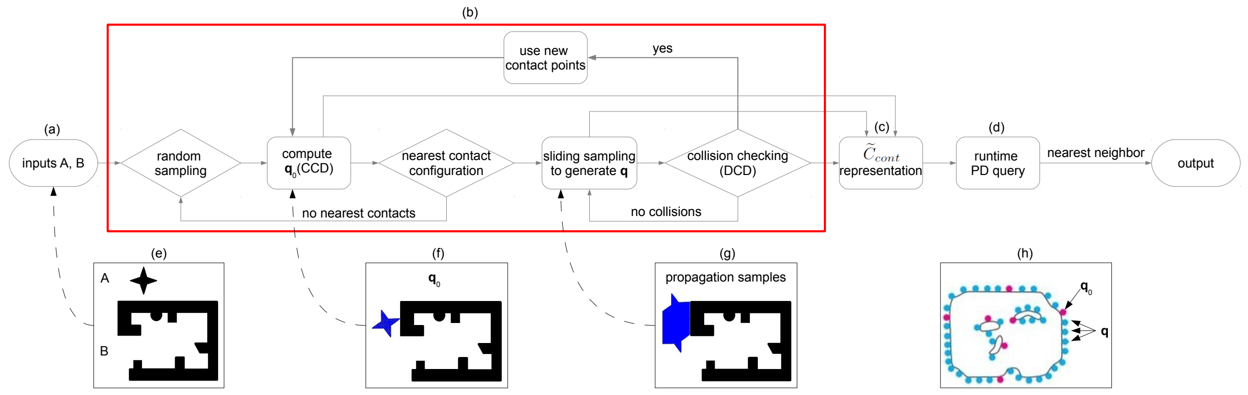


Fig. 1. The offline computation pipeline and the run-time phase of our algorithm. Given two input objects in (a), the precomputation algorithm in (b) performs the propagation sampling to efficiently generate an approximation to the contact space ($\widetilde{\mathcal{C}}_{\text{cont}}$) as the output (c). This approximate contact space is then used for efficient run-time PD query as shown in (d). (e) shows the shapes of the input objects. (f) and (g) show the relative transform between two input objects under the configuration $\mathbf{q}_0$ randomly generated as the propagation seed and a set of configurations generated from $\mathbf{q}_0$ using local-search based propagation, respectively. (h) is the output $\widetilde{\mathcal{C}}_{\text{cont}}$ where red points are the random seed samples and the blue points are the propagate-samples.

be intuitively defined as an average squared length of all displacement vectors between two objects.

C. Approximate $\mathcal{C}_{cont}$ Computation

In order to construct an approximation of the contact space, we perform an offline sampling in the configuration space, as shown in Figure 1. Given two input objects A and B, our method starts with a *random-sample* in the contact space. This random-sample can be generated using traditional continuous collision checking (CCD) techniques. These techniques compute the first time of collision or contact between two objects by reducing the problem to finding roots of polynomial equations corresponding to the triangle features. Continuous collision detection has been widely used for physically-based simulation [28] as well as local planning in robotics. Next, our method performs an iterative local search around the initial random-sample by sliding object A over object B's surface, and generates more samples on the contact space. We denote the samples generated on the contact space during the local search as *propagate-samples* to distinguish them from random-samples (see Figure 1(h)). The local search stops when no more propagate-samples can be generated, and we then restart the iteration with a new random-sample in the contact space. This iterative process continues until a sufficient number of samples have been generated. The random-samples and propagate-samples computed by our approach make up an approximate sample-based representation of the contact space $\mathcal{C}_{\text{cont}}$ between A and B, and we denote this approximation as $\widetilde{\mathcal{C}}_{\text{cont}}$.

D. Approximate PD Computation

Given the approximate representation of the contact space, $\widetilde{\mathcal{C}}_{\text{cont}}$, we compute the approximate global PD by performing a nearest-neighbor query in $\mathcal{C}_{\text{cont}}$. The definition of approximate penetration depth is analogous to the exact penetration depth in Equation 1:

$$\overline{\text{PD}}(A(\mathbf{q}_0), B) = \min_{\mathbf{q} \in \widetilde{\mathcal{C}}_{\text{cont}}} \text{dist}(\mathbf{q}_0, \mathbf{q}), \qquad (2)$$

where the domain for $\mathbf{q}$ is restricted to $\widetilde{\mathcal{C}}_{\text{cont}}$. The accuracy of $\overline{\text{PD}}$ is governed by the accuracy of $\widetilde{\mathcal{C}}_{\text{cont}}$ with respect to $\mathcal{C}_{\text{cont}}$. Given a query configuration $\mathbf{q}_0$, we perform a nearest-neighbor search to find the configuration $\mathbf{q}_c$ that is closest to the decision boundary $\widetilde{\mathcal{C}}_{\text{cont}}$. Finally, the distance between $\mathbf{q}_0$ and $\mathbf{q}_c$ is computed using an appropriate distance metric $\text{dist}(\cdot, \cdot)$ and the result is an approximation of the exact PD value. As mentioned in Section III-B, we use the object norm as the distance metric.

IV. Contact Space Propagation Sampling

In this section, we present our contact space propagation sampling algorithm that computes an offline approximation of the contact space. Our sampling algorithm is an iterative algorithm. During each iteration, we start from a random-sample in the contact space, and then perform a local search around this initial sample to generate more propagate-samples on the contact space. Once the local search stops, we repeat the iterative step with a new random-sample. This iterative process continues until a sufficient number of samples have been generated. The random-sample on the contact space during each iteration is computed by first generating two samples in the configuration space, one in collision-free space and the other in the in-collision space. We join those samples by a straight line in the configuration space and find its intersection with the contact space. This reduces to computing the first time of contact between a collision-free and an in-collision configuration, which corresponds to a CCD query. The resulting sample on the contact space is the random-sample used during this iteration.

Ideally, the local search procedure should run many steps and generate sufficient numbers of propagate-samples, in order to cover a large portion of the contact space. This coverage is important for the efficiency of the sampling algorithm, because the generation of a random-sample requires the expensive CCD query. This query is more expensive than the generation of a propagate-sample that only needs to perform the DCD (discrete collision detection) query. If the local search can generate a high number of propagate-samples, it amortizes the computational cost of generating a random-sample over a high number of propagate-samples, and improves the efficiency of our precomputation step. Our main goal is to design a fast and effective local search algorithm that can compute the propagate-samples quickly. To that end, we perform a breadth-first propagation on the contact space, starting from the random-sample. The breadth-first propagation maintains a queue (we call it the *propagate-queue*) of contact samples. During each step of this propagation, we pop one contact sample $\mathbf{q}$ from this queue, and then slide object $A(\mathbf{q})$ over the surface of object B in different directions along its boundary. This propagation step results in a set of new samples $\{\mathbf{q}'\}$ around $\mathbf{q}$, as shown in Figure 1. Next, we perform collision checking for these samples $\{\mathbf{q}'\}$, and only add the collision-free samples into the propagate-queue. The breadth-first search is repeated until the queue is empty.

A new sample $\mathbf{q}'$ is propagated from a contact sample $\mathbf{q}$ and is added into the propagate-queue only if it is collision-free. Otherwise, the sample would be discarded and the actual execution of the local search's propagation may be interrupted, as shown in Figure 2. To overcome this challenge, we classify the local search process into two cases: the *boundary configuration case* when $\mathbf{q}' \in \mathcal{C}_{\text{cont}}$, and the *internal configuration case* when $\mathbf{q}' \in \mathcal{C}_{\text{obs}}$. In the internal configuration case, we resume the local search computation according to objects' contact features (i.e., local vertices, edges, faces). This formulation greatly improves the efficiency of the local search. The overall local search computation algorithm is shown in Algorithm 1.

Algorithm 1: Local search for propagate-samples

```
input : Two objects $A$ and $B$, an initial random-sample $\mathbf{q}^r$,
        the search step size $d$
output: A set of propagate-samples $S$ from $\mathbf{q}^r$
/* Initialize final result set */
1  $S \leftarrow \emptyset$ ;
/* Initialize a propagate-queue $Q$ */
2  $p_A$, $p_B \leftarrow$ contact points of $A(\mathbf{q}^r)$ and $B$ ;
3  $\theta \leftarrow$ angles between contact normals ;
4  $Q \leftarrow \{(\mathbf{q}^r, p_A, p_B, \theta)\}$ ;
5  while $Q \neq \emptyset$ do
6      $(\mathbf{q}, p_A, p_B, \theta) \leftarrow$ pop($Q$) ;
7      $N \leftarrow B$'s vertices at a step of $d$ away from $p_B$ ;
8      for $p'_B \in N$ do
9          $(\mathbf{q}', p_A, p'_B, \theta) \leftarrow \mathcal{T}(\mathbf{q}, p_A, p_B, \theta)$ ;
10         if isCollision($\mathbf{q}'$) then
               /* Internal configuration case */
11             Compute the critical $\mathbf{q}^c$ between $\mathbf{q}$ and $\mathbf{q}'$ ;
12             $M \leftarrow$ contact pairs other than $(p_A, *)$ between
               $A(\mathbf{q}^c)$ and $B$ ;
13             for $(p^c_A, p^c_B) \in M$ do
14                 $\theta^c \leftarrow$ angles between contact normals at $p^c_A$
                   and $p^c_B$ ;
                   /* Check whether $\mathbf{q}^c$ is close to
                      previous samples */
15                 if kdTreeTest($S$,$\mathbf{q}^c$,$r$) = true then
16                     continue ;
17                 $Q \leftarrow Q \cup \{(\mathbf{q}^c, p^c_A, p^c_B, \theta^c)\}$ ;
18         else
               /* Boundary configuration case */
19             $Q \leftarrow Q \cup \{(\mathbf{q}', p_A, p'_B, \theta)\}$ ;
20     $S \leftarrow S \cup \{\mathbf{q}\}$ ;
```

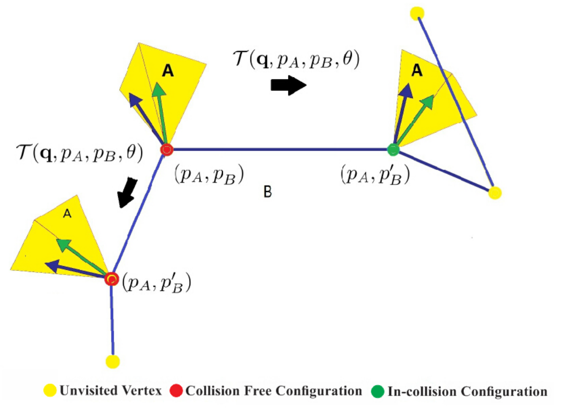
A. Boundary Configuration Case

In the boundary configuration case, each step of the local search is a standard propagation step from a contact sample $\mathbf{q}$. We first compute the contact pair (p_A, p_B) between $A(\mathbf{q})$ and B, where p_A and p_B are two contact points on objects A and B. The points p_A and p_B belong to two different objects but overlap with each other, just touching at the configuration $\mathbf{q}$. We also compute the angle θ between the normal vectors at p_A and p_B. Next, we slide the object A over the surface of object B with a distance d, which can be any value less than the edge length of triangular mesh(since we generate the samples on the mesh of object). For our case, we we use the edge lengths of the fixed objects as the sliding step, and thus the propagate-samples all land in the vertices of object B. During the sliding movement, the contact point on A remains unchanged as p_A, and the contact point on B moves from p_B to p'_B (see Figure 2). Now A and B touch at the new contact point pair (p_A, p'_B). We further rotate the object A such that the angle between the two objects' contact normals remains to be θ (as shown in Figure 2). In this way, we compute a new configuration $\mathbf{q}'$, which can be specified using p_A, p'_B and θ for 2D objects. Similar propagation procedure can also be defined for 3D objects, whose configuration space has 6 dimensions.

We represent the sliding movement from $\mathbf{q}$ to $\mathbf{q}'$ as a transition function $(\mathbf{q}', p_A, p'_B, \theta) = \mathcal{T}(\mathbf{q}, p_A, p_B, \theta)$. The new generated configuration $\mathbf{q}'$ is pushed into the propagate-queue for future propagation-sample computations. This sliding procedure is executed along different directions on the surface of object B around p_B, and results in a set of new configurations $\{\mathbf{q}'\}$ spreading over the neighborhood of $\mathbf{q}$ on the contact space. The collision-free samples in $\{\mathbf{q}'\}$ are located on the contact space and we add them directly into the propagate-queue. For in-collision samples in $\{\mathbf{q}'\}$, we treat them as the internal configuration case and stop propagation.

In the above description, we restrict all new propagate-samples $\{\mathbf{q}'\}$ to have the same angle θ between the two objects' contact normals. This simple heuristic greatly increases the probability that a new sample $\mathbf{q}'$ will be collision-free. It is based on the assumption that the surface curvature around the neighborhood of p_B is roughly constant. Therefore, a configuration $\mathbf{q}'$ with the same relative angle as $\mathbf{q}$ should have a high probability to be collision-free after the sliding movement.

The parameter d in a boundary configuration propagation step determines the step-size of the propagation. Its value varies during the local search process. In particular, d is inversely proportional to the relative scale of A with respect to B, and it is also related to the surface curvature at p_B.



$\mathcal{T}(\mathbf{q}, p_A, p_B, \theta)$
A
$\mathcal{T}(\mathbf{q}, p_A, p_B, \theta)$
A
(p_A, p_B)
B
(p_A, p'_B)
A
(p_A, p'_B)
● Unvisited Vertex ● Collision Free Configuration ● In-collision Configuration

Fig. 2. An example of the contact sample propagation between an object A (the yellow tetrahedron) and an object B (the blue polygon). The object A is initially at configuration $\mathbf{q}$ where the contact points between two objects are p_A and p_B and the angle between contact normals is θ. During the propagation, the object A moves along one edge of the object B's mesh and finally contacts with object B at p'_B, a neighbor vertex of p_B. Object A always has point p_A as its contact point.

B. Internal Configuration Case

In this case, $\mathbf{q}$ is inside $\mathcal{C}_{\text{obs}}$, i.e. inside the C-obstacle space. This can happen when two objects are very close in size or the surface of B is 'bumpy', i.e., the curvature changes dramatically over the surface. In this case, the propagation search step usually explores only a few steps because A and B will collide even when A only slides a small step over B's surface. In an extreme situation, every local search returns no propagation-samples and all samples generated on the contact surface are random-samples. This will result in very slow sampling procedure, and these samples cannot be evenly distributed over the contact space.

Our solution for the internal configuration case exploits the contact features, and is based on the following property of the sliding movement:

Theorem 1: Suppose one step of slide moves a contact sample $\mathbf{q}^0$ into an in-collision sample $\mathbf{q}^1$. The transition function is

$$(\mathbf{q}^1, p_A, p_B^1, \theta) = \mathcal{T}(\mathbf{q}^0, p_A, p_B^0, \theta), \tag{3}$$

where (p_A, p_B^0) is the contact pair at $\mathbf{q}^0$, (p_A, p_B^1) is the contact pair at $\mathbf{q}^1$, and θ is the angle between contact boundary configurations. On the resulting sliding trajectory, there exists one configuration $\mathbf{q}^t$ such that $A(\mathbf{q}^t)$ and B are in contact, but have at least one additional contact point other than p_A. Here we assume the sliding movement is parameterized by $t \in [0, 1]$. We call the configuration $\mathbf{q}^t$ the critical configuration.

The proof is given in [36]. Based on this theorem, we resume the propagation at an in-collision sample $\mathbf{q}'$ by first computing the critical configuration $\mathbf{q}^c$ between $\mathbf{q}$ and $\mathbf{q}'$. $\mathbf{q}^c$ has more than one contact point, and we denote the set of contact points other than p_A as M. For each contact point p_A^c in M, we continue the local search step by changing the contact point on A that remains unchanged during the movement from p_A to p_A^c. In particular, we compute the contact pair (p_A^c, p_B^c) and θ^c, the angle between the contact boundary configurations at p_A^c and p_B^c. Object A now starts sliding from the contact point p_A^c and the corresponding transition function is $\mathcal{T}(\mathbf{q}^c, p_A^c, p_B^c, \theta^c)$. Figure 3 illustrates this process.

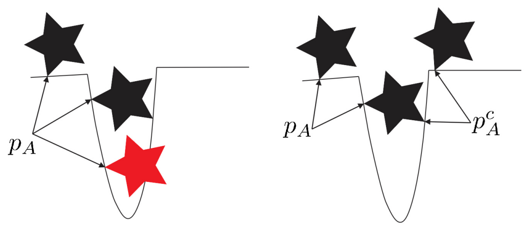


Fig. 3. Objects with highly varying curvatures where the sliding movements can result in in-collision configurations and the early termination of the propagation procedure. After applying the change of contact points, the local search can escape from the 'bumpy' surface region. In the left figure, the star shaped object A moves over the surface B, and the red star denotes the sample configuration corresponding to the internal configuration case. In the right figure, we compute the critical configuration $\mathbf{q}^c$ and find a new contact point p_A^c.

To avoid generating repeated samples during the internal configuration case, we use a kd-tree to check the new contact configuration $\mathbf{q}^c$ and use it as the new propagation seed only if it is not close to any existing samples, as shown in the line 15 in Algorithm 1.

C. Run-time PD Queries

In the PD query stage, we use the contact configurations generated during the precomputation. Given a query configuration $\mathbf{q} \in \mathcal{C}_{obs}$, we use nearest-neighbor query to find two contact configurations $\mathbf{q}_c^0$ and $\mathbf{q}_c^1$ that are closest to $\mathbf{q}$ and incident to a same triangle on the fixed object. Next, we compute a configuration $\mathbf{q}_c$ which is a linear combination of these two contact configurations, and the line connecting $\mathbf{q}^2$ an $\mathbf{q}$ is perpendicular to the line connecting $\mathbf{q}_c^0$ and $\mathbf{q}_c^1$:

$$\mathbf{q}^2 = (1-\rho)\mathbf{q}_c^0 + \rho\mathbf{q}_c^1; \quad \text{and} \quad (\mathbf{q}^2 - \mathbf{q}) \perp (\mathbf{q}_c^1 - \mathbf{q}_c^0). \tag{4}$$

We then perform a linear search from $\mathbf{q}^2$ along directions of $\mathbf{q} - \mathbf{q}^2$ until we find a collision-free configuration $\mathbf{q}_c^2$. The nearest-neighbor query is performed based on the $\text{dist}()$ metrics listed in Section III. Finally, we have to compare the distance of $\mathbf{q}^2$ with those of $\mathbf{q}_c^0$, $\mathbf{q}_c^1$, and choose the smallest one as the PD value of $\mathbf{q}$.

V. Performance and Comparison

In this section, we highlight the performance of our propagation sampling based precomputation and the run-time PD query on a set of challenging 3D benchmarks as shown in Figure 4. We use PQP query package to perform all the collision tests used in our framework. For PD query, we investigate both the translational and generalized PD, using the distance metric as mentioned in Section III.

We implement our algorithm in C++ on an Intel Core i7 CPU running at 3.30GHz with 16GB of RAM on Window 7 (64-bits) PC. All the performance and timing results are generated using a single core. Our precomputation algorithm can be easily parallelized on a multi-core PC because both the random-samples and the propagate-samples on the contact space can be generated in parallel.

#propagations	30,000	50,000	80,000	100,000	#samples
Donut	314	551	693	891	79
CAD1	421	569	745	883	411
CAD2	390	581	737	923	323
Teeth	520	892	1,455	1,621	1231
Dragon	590	940	1,401	1,632	2997
Buddha	603	875	1,337	1,501	1820

TABLE I

PERFORMANCE OF THE PRECOMPUTATION COST FOR GENERAL PD ON DIFFERENT BENCHMARKS. WE VARY THE NUMBER OF PROPAGATION STEPS (I.E., THE NUMBER OF RANDOM-SAMPLES) FROM $30,000$ TO $100,000$, AND THE CORRESPONDING TIME COSTS (IN SECONDS) ARE SHOWN IN THE FIRST FOUR COLUMNS. WE ALSO PROVIDE THE AVERAGE NUMBER OF SAMPLES PER PROPAGATION STEP IN THE COLUMN #SAMPLES.

#propagations	30,000	50,000	80,000	100,000	#samples
Donut	156	193	344	423	63
CAD1	159	243	387	659	329
CAD2	163	247	401	613	341
Teeth	155	471	912	973	768
Dragon	333	752	941	1,429	667
Buddha	346	771	1,017	1,918	751

TABLE II

PERFORMANCE OF THE PRECOMPUTATION COST FOR TRANSLATIONAL PD ON DIFFERENT BENCHMARKS. WE VARY THE NUMBER OF PROPAGATION STEPS (I.E., THE NUMBER OF RANDOM-SAMPLES) FROM $30,000$ TO $100,000$, AND THE CORRESPONDING TIME COSTS (IN SECONDS) ARE SHOWN IN THE FIRST FOUR COLUMNS. WE ALSO PROVIDE THE AVERAGE NUMBER OF SAMPLES PER PROPAGATION STEP IN THE COLUMN #SAMPLES.

A. *Performance*

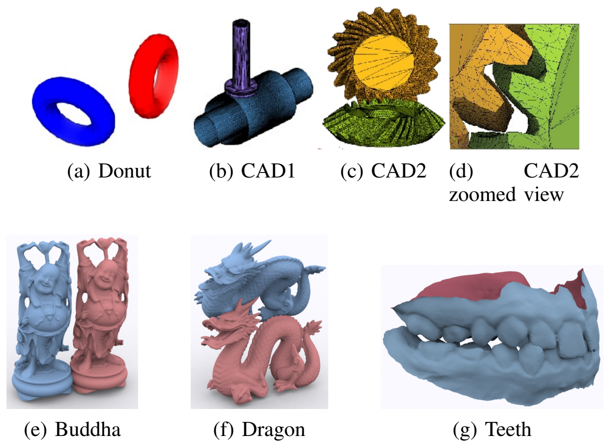


Fig. 4. The benchmarks we used to investigate the PD performance: (a) Donut, each with 576 triangles; (b) CAD1 with about 10K triangles each; (c) CAD2 with about 12K triangles each; (d) a zoomed view of CAD2 is also provided to show the complexity of this benchmark; (e) Buddha with 1M triangles each; (f) Dragon with with 230K triangles each; (g) Teeth models with about 40K triangles each.

Time Cost: We now highlight the performance of both the off-line precomputation and run-time PD queries. Table I and Table II show the time costs of the precomputation phase for general PD and translational PD respectively, given different number of propagation steps. We use at most 100,000 propagation steps, and each step will generate one random-sample and a set of propagate-samples extended from the random-sample. We can see that the time cost for a single propagation step increases while more samples are generated, and this is due to the fact that while more samples are generated, it is more difficult to find unrepeated new samples in the contact space. Table I and Table II also provide the number of samples generated for each propagation step, which is averaged over all the 100,000 propagation steps. Table III provides our method's run-time cost for a single PD query, which is computed as the average time cost after computing 1,000 randomly generated in-collision queries.

Convergence: We investigate the convergence of our sampling algorithm by varying the number of samples and evaluating its benefits in terms of approximating the contact space. Since it is difficult to obtain the ground truth for the general PD, *the investigation is only for the translational PD*. We use the result from the Minkowski sum based approach [31] as the ground truth of the contact space, and then we compute the error between the PD results from the ground truth and the PD results provided by our propagation sampling approach.

As we generate more random-samples and propagate-samples, our precomputation algorithm provides a better approximation of the contact space. To evaluate the approximation quality of the contact space, we measure the number of vertices in the original models that have been visited during the propagation as contact points (e.g., p_A and p_B in Figure 2). This is due to the fact that the denser the contact space is sampled, the more vertices should be visited as contact points during the sampling process. The measurement result is shown in Figure 5, and we can observe that for all benchmarks, the propagation procedure convergence after generating 10 million samples, and a high quality approximation to the contact space is achieved.

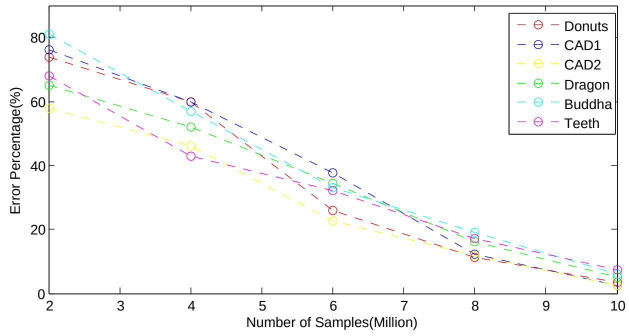


Fig. 5. Convergence analysis of our precomputation algorithm for general PD. The x-axis corresponds to the number of samples (both random and propagation). The y-axis corresponds to our method's error, which decreases while more samples are generated. In all cases, our algorithm converges quickly after 10 million samples.

B. *Comparison with Prior Methods*

We compare the performance of our algorithm with two recent methods that can compute translational and generalized PD. One is a global approach [10] that uses active learning to generate the samples in the entire configuration space, and then computes the contact space approximation as

	Donut	CAD1	CAD2	Dragon	Teeth	Buddha
Our (PD_g)	0.93	0.95	1.03	1.18	0.98	1.31
Our(PD_t)	0.81	0.85	1.13	1.06	1.01	1.11
Active [10]	0.95	1.41	1.32	1.43	0.92	1.48
Local [13]	6.56	78.3	54.4	132	111	153

TABLE III

COMPARISON OF THE RUN-TIME COST FOR A SINGLE PD QUERY (IN MS) BETWEEN OUR METHOD AND TWO PRIOR METHODS: ACTIVE LEARNING [10] AND LOCAL OPTIMIZATION [13], [4]. OUR METHOD USES $100,000$ RANDOM-SAMPLES FOR PRECOMPUTATION, AND THE TIME COST IS AVERAGED OVER $1,000$ RANDOMLY GENERATED IN-COLLISION QUERIES.

the surface separating collision-free and in-collision samples. Its approximation of the contact surface is relatively smooth due to its leverage of SVMs and hence this method may not provide accurate PD results for query configurations with small penetrations. Overall, this approach provide a conservative bound on the PD. The second method uses local optimization [13], [4] to compute translational and generalized PD. It starts from a good initial guess to the PD result and optimizes the computation based on appropriate metric. Its accuracy depends on the initial guess and it may get stuck in a local minimum. Unlike our method or the active learning approach, this method has no precomputation. Therefore this approach can also be used for PD computation involving deformable models. Recently, Kim et al. [34] presented a hybrid approach that combines the global method [10] with the local method [13] to overcome some of these problems.
Runtime Performance: We compare the performance of runtime query of our approach with these methods, and the result is shown in Table III, for both general and translational PD formulations. We observe that our approach is significantly faster than other two approaches, because our method can generate samples more evenly and densely distributed in the contact space, and thus query configuration can easily find a nearest-neighbor contact configuration in fewer iterations.
Accuracy of PD computation: In our experiments, our method quickly generates more than 1 million samples in 10-12 minutes with less than 5 MB storage. The error of our online PD query is about 5% of the actual PD value for both the translational and general PD, while this value is 15-20% for [10] and 10% for [13], [4]. The main reason for this improvement is because our method searches over the entire contact space globally while the local optimization methods [13], [4] search along the gradient direction. The gradient search strategy can only generate suboptimal results, especially for complex models. Our approach also outperforms [10], [34] which approximates the contact space by active learning, because it can provide a more accurate approximation of the contact space. Overall, our approach is more accurate than prior PD methods, and detailed accuracy comparisons are provided in [36].

VI. ANALYSIS

In this section, we evaluate some properties of the propagation sampling algorithm used in our precomputation phase. In particular, we discuss 1) the bounds on the time complexity of our precomputation scheme, and 2) why our approach can generate samples with better distribution in the contact space than pure random sampling.

A. Time Complexity

For the time complexity analysis, we assume the object A can only perform translational movements and has m vertices, and object B is a connected mesh with n vertices, where $m \ll n$. We also denote T_{DCD} and T_{CCD} as the time costs for one continuous collision checking and one discrete collision checking respectively. On average, $T_{CCD} \approx 7.46 T_{DCD}$ in our 10 thousand random tests. Then we have the following estimation for the precomputation's time complexity:

Theorem 2: The precomputation's time complexity has a lower bound of $T_{CCD} + (n-1)T_{DCD}$, and has an upper bound of $n \lg(n) T_{CCD}$.

The proof for this theorem is in [36]. The two bounds estimated in this theorem are conservative, but intuitively describes the performance of our precomputation scheme. In practice, the running time of our algorithm is much lower than the upper bound estimated, because for each propagation iteration started from a random-sample, we can generate a large number of propagate-samples due to our special treatment to the internal configuration cases in Section IV.

B. Sample Distribution

We next show that our method can generate samples evenly distributed over the contact space. First, the propagation movement has the following property:

Theorem 3: **Commutativity:** Suppose the object A slides along the surface of the object B according to the transition function $(\mathbf{q}^1, p_A, p_B^1, \theta) = \mathcal{T}(\mathbf{q}^0, p_A, p_B^0, \theta)$, where the transition function is as described in Equation 3. $\mathbf{q}^0$ and $\mathbf{q}^1$ are object A's configurations before and after the transition, and p_B^0 and p_B^1 are the contact points on the object B for these two configurations. Then the inverse slide can be formulated by the transition function $(\mathbf{q}^0, p_A, p_B^0, \theta) = \mathcal{T}(\mathbf{q}^1, p_A, p_B^1, \theta)$.

Next, we show that the samples generated by our propagation scheme will have no duplications:

Theorem 4: **Uniqueness:** Any two sets of samples generated by different propagation iterations will have no overlaps. The proofs for these theorems are in [36]. Overall, our propagation sampling method has three benefits:

- **Efficiency** Theorems 3 and 4 imply that no duplicated samples would be generated during the propagation. As a result, the algorithm will not waste time on generating redundant samples, and can make steady progress toward constructing a more precise representation of the contact space while generating more and more samples. These properties result in the error reduction shown in Figure 5. In addition, these two theorems also guarantee the even distribution of samples over the surface of the contact space.

- **Accuracy** Our propagation steps directly slide the moving object over the surface of the fixed objects. As a result, every generated sample locates exactly on the contact space, and this results in the high accuracy of the PD results.

VII. Limitations, Conclusions and Future Work

We present a new PD approximation algorithm between general 3D models. We compute an approximation of contact space using propagation sampling. The propagation sampling scheme improves the accuracy of the approximation and is much faster as compared to only using random samples. We highlight the performance on many complex 3D models and highlight the benefits in terms of runtime performance and accuracy.

Our approach has some limitations. If the contact space has very narrow components, our sampling approach may miss them and thereby affect the accuracy of PD computations. It is possible that propagation sampling may not find samples during the local search computation. We would like to investigate the use of narrow-passage algorithms in sample-based motion planning to improve the performance. It would be useful to improve the performance of propagation-sampling by utilizing the local curvature of the surface to choose appropriate directions. We would like to evaluate the performance on complex and articulated models, and also integrate with dynamics simulation and motion planning algorithms.

References

[1] D. Baraff and A. Witkin, *Physically Based Modeling*. ACM SIGGRAPH Course Notes, 2001.

[2] L. Zhang, Y. J. Kim, G. Varadhan, and D. Manocha, "Generalized penetration depth computation," *Computer Aided Design*, vol. 39, no. 8, pp. 625–638, 2007.

[3] D. Wang, S. Liu, X. Zhang, and J. Xiao, "Configuration-based optimization for six degree-of-freedom haptic rendering for fine manipulation," *IEEE Transactions on Haptics*, vol. 5, no. 4, pp. 332–343, 2012.

[4] C. Je, M. Tang, Y. Lee, M. Lee, and Y. J. Kim, "Polydepth: Real-time penetration depth computation using iterative contact-space projection," *ACM Transactions on Graphics*, vol. 31, no. 1, pp. 5:1–5:14, Feb. 2012.

[5] M. Koval, N. Pollard, and S. Srinivasa, "Manifold representations for state estimation in contact manipulation," in *International Symposium on Robotics Research*. Springer, 2013.

[6] E. Catto, "Box2D: A 2d physics engine for games," http://box2d.org, 2010.

[7] E. Coumans, "Bullet physics library," http://bulletphysics.org, 2010.

[8] B. Heidelberger, M. Teschner, R. Keiser, M. Mller, and M. H. Gross, "Consistent penetration depth estimation for deformable collision response," in *International Fall Workshop on vision, modeling and visualization*, 2004, pp. 339–346.

[9] S. Redon and M. C. Lin, "A fast method for local penetration depth computation," *Graphical Tools*, vol. 8, no. 1, pp. 63–70, 2006.

[10] J. Pan, X. Zhang, and D. Manocha, "Efficient penetration depth approximation using active learning," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 191:1–191:12, Nov. 2013.

[11] Y. J. Kim, M. A. Otaduy, M. C. Lin, and D. Manocha, "Fast penetration depth computation for physically-based animation," in *Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002, pp. 23–31.

[12] J.-M. Lien, "A simple method for computing Minkowski sum boundary in 3D using collision detection," in *Algorithmic Foundation of Robotics VIII*, ser. Springer Tracts in Advanced Robotics. Springer Berlin / Heidelberg, 2009, vol. 57, pp. 401–415.

[13] M. Tang and Y. J. Kim, "Interactive generalized penetration depth computation for rigid and articulated models using object norm," *ACM Transactions on Graphics*, vol. 33, no. 1, pp. 1:1–1:15, Feb. 2014.

[14] G. Varadhan, Y. J. Kim, S. Krishnan, and D. Manocha, "Topology preserving approximation of free configuration space," in *Proceedings of International Conference on Robotics and Automation*, 2006, pp. 3041–3048.

[15] M. Sharir, "Algorithmic motion planning," in *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. Boca Raton, FL: CRC Press LLC, 2004, ch. 47, pp. 1037–1064.

[16] E. Sacks and C. Bajaj, "Sliced configuration spaces for curved planar bodies," *International Journal of Robotics Research*, vol. 17, pp. 639–651, 1997.

[17] L. J. Guibas and R. Seidel, "Computing convolutions by reciprocal search," *Discrete & Computational Geometry*, vol. 2, no. 1, pp. 175–193, 1987.

[18] T. Lozano-Pérez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. C-32, no. 2, pp. 108–120, 1983.

[19] X. Ji and J. Xiao, "On random sampling in contact configuration space," in *Workshop on Algorithmic Foundations of Robotics*, 2000.

[20] L. He and J. V. D. Berg, "Efcient exact collision-checking of 3-d rigid body motions using linear transformations and distance computations in workspace," in *IEEE International Conference on Robotics and Automation*, 2014.

[21] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based prm for 3d workspaces," in *Workshop on the Algorithmic Foundations of Robotics*, 1998, pp. 155–168.

[22] O. Salzman, M. Hemmer, B. Raveh, and D. Halperin, "Motion planning via manifold samples," *Algorithmica*, vol. 67, no. 4, pp. 547–565, 2013.

[23] G. van den Bergen, "Proximity queries and penetration depth computation on 3D game objects," in *Game Developers Conference*, 2001.

[24] P. K. Agarwal, L. J. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir, "Computing the penetration depth of two convex polytopes in 3D," in *Proceedings of Scandinavian Workshop on Algorithm Theory*, 2000, pp. 328–338.

[25] Y. J. Kim, M. C. Lin, and D. Manocha, "DEEP: Dual-space expansion for estimating penetration depth between convex polytopes," in *Proceedings of International Conference on Robotics and Automation*, 2002, pp. 921–926.

[26] E. Guendelman, R. Bridson, and R. Fedkiw, "Nonconvex rigid bodies with stacking," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 871–878, 2003.

[27] M. Tang, M. Lee, and Y. J. Kim, "Interactive hausdorff distance computation for general polygonal models," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 74:1–74:9, 2009.

[28] M. Tang, D. Manocha, M. A. Otaduy, and R. Tong, "Continuous penalty forces," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 107:1–107:9, 2012.

[29] R. Weller and G. Zachmann, "Inner sphere trees for proximity and penetration queries." in *Robotics: Science and Systems*, vol. 2, 2009.

[30] B. Wang, F. Faure, and D. K. Pai, "Adaptive image-based intersection volume," in *Proceedings of SIGGRAPH*, 2012, pp. 97:1–97:9.

[31] J.-M. Lien, "Covering Minkowski sum boundary using points with applications," *Computer Aided Geometric Design*, vol. 25, no. 8, pp. 652–666, 2008.

[32] G. Nawratil, H. Pottmann, and B. Ravani, "Generalized penetration depth computation based on kinematical geometry," *Computer Aided Geometric Design*, vol. 26, no. 4, pp. 425–443, May 2009.

[33] L. Zhang, Y. J. Kim, and D. Manocha, "A fast and practical algorithm for generalized penetration depth computation," in *Robotics: Science and Systems*, 2007.

[34] Y. Kim, D. Manocha, and Y. Kim, "Hybrid penetration depth computation using local projection and machine learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[35] K. Kazerounian and J. Rastegar, "Object norms: A class of coordinate and metric independent norms for displacement," in *ASME Design Technical Conference*, 1992, pp. 271–275.

[36] L. He, J. Pan, D. Li, and D. Manocha, "Efficient penetration depth computation between rigid models using contact space propagation sampling," *CoRR*, vol. abs/1511.03999, 2015. [Online]. Available: http://arxiv.org/abs/1511.03999