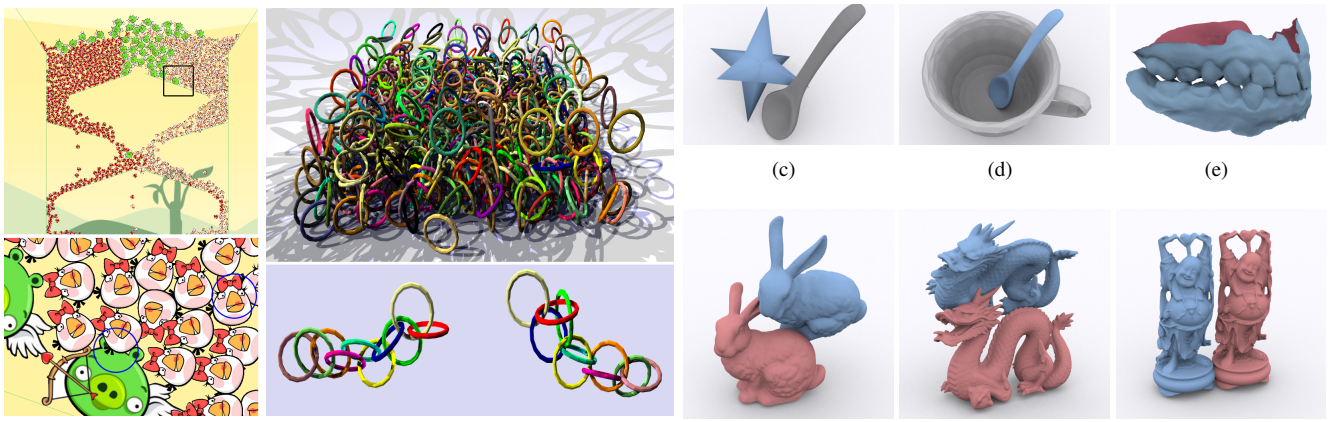
Efficient Penetration Depth Approximation using Active Learning

Jia Pan Xinyu Zhang Dinesh Manocha *
University of North Carolina at Chapel Hill



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 1: *Our algorithm computes a global penetration depth between overlapping non-convex and non-manifold objects. (a) Dynamic simulation of angry bird characters falling into a complex chute in Box2D physics engine; (b) rainfall of* $1,000$ *rings in Bullet physics engine; (c) a star and a spoon; (d) a spoon and a cup; (e) multiple contacts between upper and lower teeth (each has more than* $40,000$ *triangles). (f-h) benchmarks consisting of complex models (bunny, dragon and Buddha models have* $70K$*,* $230K$ *and* $1M$ *triangles, respectively). Our PD algorithm takes less than* $0.1 \sim 2$ *milliseconds, with less than 2-3% relative error, for each pair of overlapping objects.*

Abstract

We present a new method for efficiently approximating the *global penetration depth* between two rigid objects using machine learning techniques. Our approach consists of two phases: offline learning and performing run-time queries. In the learning phase, we precompute an approximation of the *contact space* of a pair of intersecting objects from a set of samples in the configuration space. We use active and incremental learning algorithms to accelerate the precomputation and improve the accuracy. During the run-time phase, our algorithm performs a nearest-neighbor query based on translational or rotational distance metrics. The run-time query has a small overhead and computes an approximation to global penetration depth in a few milliseconds. We use our algorithm for collision response computations in Box2D or Bullet game physics engines and complex 3D models and observe more than an order of magnitude improvement over prior PD computation techniques.

CR Categories: I.3.3 [Computer Graphics]: Computational Geometry and Object Modeling—Geomtric algorithms, languages and systems

Keywords: Contact Space, Penetration Depth, Support Vector Machine, Active Learning, Dynamic Simulation

Links: DL PDF WEB

*e-mail: {panj,zhangxy,dm}@cs.unc.edu

1 Introduction

Measuring the extent of inter-penetration between two intersecting objects is an important problem in physically-based simulation and geometric computing. A widely used metric to quantify inter-penetration is the *penetration depth* (PD), which is defined as the minimum amount of motion transformation required to separate two intersecting objects. This transformation may correspond to only translation (translational PD) or both translation and rotation (generalized PD). PD computation is widely used for contact resolution in dynamic simulation [Baraff and Witkin 2001], tolerance verification for virtual prototyping [Kim et al. 2002a], force computation in haptic rendering [Wang et al. 2012b], motion planning in robotics [Zhang et al. 2007b], etc. Many well known game physics engines such as Box2D [Catto 2010] and Bullet [Coumans 2010] also perform PD computation for collision response.

The time complexity of exact PD computation in 3D can be as high as $\mathcal{O}(m^3n^3)$ for translational PD and $\mathcal{O}(m^6n^6)$ for generalized PD, where m and n are the number of triangles in two non-convex input models. Given the high combinatorial complexity of exact PD computation, many approximate algorithms have been proposed. At a broad level, prior methods can be classified into techniques based on local feature analysis, constrained optimization, distance fields, convex-decomposition, and point-based approximations. Many of these algorithms compute local PD based on overlapping features, and may not provide a reliable solution for general non-convex models (see Figure 2). The accuracy of local PD algorithms varies based on the relative configuration of the two objects and on the heuristics used to estimate the extent and direction of penetration [Heidelberger et al. 2004; Tang et al. 2012]. Methods based on convex decomposition or point-based approximations can compute a more reliable measure of global penetration, but they are too slow for interactive applications.

Main Results: We present a novel algorithm to approximate global PDs between rigid objects for interactive applications using machine learning classification techniques. Our approach is applicable to all rigid models and can compute translational and gener-

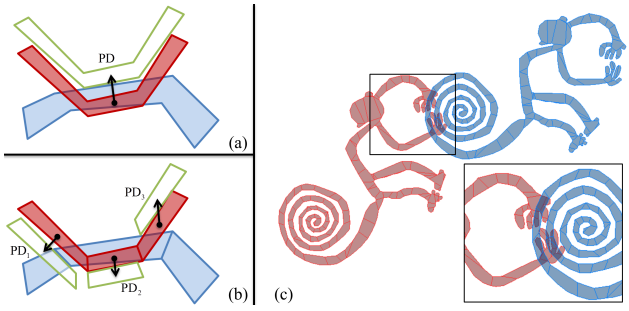


Figure 2: *(a) Global translational PD between non-convex models (shown in red and blue). (b) Local translational PD corresponding to each isolated contact region (shown with different arrows), which does not separate the overlapping objects. (c) In this Box2D simulation, local PD algorithms cannot separate the non-convex objects with deep inter-penetration.*

alized PD (see Section 3). The main idea is to precompute an approximation of the contact space of two overlapping objects by generating configuration space samples and computing a decision function using support vector machines (see Section 4). In order to reduce the number of samples and improve the accuracy of this representation, we use active and incremental machine learning techniques. At runtime, our algorithm performs a nearest-neighbor query between a given configuration and precomputed contact space approximation (see Section 5). As compared to prior techniques, our approach offers the following benefits:

- The overall algorithm is general and directly applicable to complex non-convex and non-manifold models.
- This formulation can be used to compute translational *and* generalized PD between 2D and 3D objects.
- The use of active learning can drastically reduce the number of samples and improves the convergence rate of our approximation scheme (see Section 6).
- The runtime query has a small overhead (a few milliseconds) and can be used for interactive applications (see Section 7).

We highlight the performance of our algorithm on complex non-convex models with multiple contacts. In practice, we are able to compute approximate PD with less than 2-3% relative error by using a few thousand samples during the learning phase (see Section 7). We also use our PD algorithm to compute collision response between non-convex models in Box2D and Bullet physics engines. As compared to prior global PD algorithms, we observe more than an order of magnitude improvement in runtime performance. To the best of our knowledge, this is the first practical algorithm for reliable PD computation between non-convex models for interactive applications.

2 Related Work

There is extensive work on PD computation in computer graphics, geometric modeling, haptics, and robotics. We give a brief overview of exact and approximate computation algorithms.

For convex polytopes, exact translational PD can be computed using the Minkowski sum [van den Bergen 2001; Agarwal et al. 2000; Kim et al. 2002a]. For non-convex objects, the PD can be computed using a combination of convex decomposition, pairwise Minkowski sums, and union computation [Kim et al. 2002b]. these algorithms are applicable to closed polyhedral shapes. The computational complexity of union computation is high and it can be approximated using rasterization hardware [Kim et al. 2002b].

Most practical techniques for translational PD compute local PD or some approximation of global PD. Local PD algorithms only take account of local overlapping features (vertices, edges and faces), and compute a transformation to separate those features [Guendelman et al. 2003; Redon and Lin 2006; Lien 2009; Tang et al. 2009; Tang et al. 2012]. Intersection volume and its derivative can also be used for volume-based repulsion in [Wang et al. 2012a]. Distance fields are also used for local translational PD computation [Heidelberger et al. 2004] and can be computed in realtime using GPUs. Point-based Minkowski sum approximation [Lien 2008] can also be used to compute global translational PDs.

Exact generalized PD can be computed by constructing the exact contact space and then searching the contact space for the closest point to a given query [Zhang et al. 2007b]. However, due to high time and storage complexity, most generalized PD algorithms use optimization-based techniques [Nawratil et al. 2009; Zhang et al. 2007a; Je et al. 2012] and compute a locally optimal solution based on local approximation of the contact space.

Machine learning techniques have been used for collision detection [Doshi et al. 2007; Pan et al. 2011]. However, these techniques cannot be used for PD computation directly. In practice, checking for collisions is much easier than computing the PD between overlapping objects.

3 Background and Overview

In this section, we introduce our notation and give an overview of our approach. We first present PD formulation in terms of configuration space and then describe our approach to computing approximate PD using active learning.

3.1 Contact Space and PD Formulation

Given two objects A and B, we denote their configuration space as $\mathcal{C}$-space. Each configuration or point in $\mathcal{C}$-space corresponds to the relative configuration (i.e., position and orientation) of A with respect to B. In the rest of the paper, we assume that A is movable and B is fixed. For 2D objects, $\mathcal{C}$-space has 2 degrees of freedom (DOF) if A can only undergo translation motion, and 3-DOF if we take account of translational and rotational motion. For 3D objects, $\mathcal{C}$-space has 3-DOF for translational motion and 6-DOF for both translational and rotational motion. $\mathcal{C}$-space is composed of two components: *collision-free* space $\mathcal{C}_{free} = \{\mathbf{q} : A(\mathbf{q}) \cap B = \emptyset\}$ and *in-collision* or *obstacle* space $\mathcal{C}_{obs} = \{\mathbf{q} : A(\mathbf{q}) \cap B \neq \emptyset\}$, where $A(\mathbf{q})$ corresponds to A located at the configuration $\mathbf{q}$.

Contact Space: It is the boundary of $\mathcal{C}_{obs}$ and is denoted as $\mathcal{C}_{cont} = \partial C_{obs}$. Intuitively speaking, a contact space corresponds to the configurations where A and B just touch each other without any penetration. Figure 3-(a) shows an example of the $\mathcal{C}$-space of two objects A and B, where $\mathcal{C}_{cont}$ is highlighted with orange curves. We use the notation $c(\mathbf{q}) \in \{-1, +1\}$ to denote the collision state of a configuration $\mathbf{q}$, i.e., $c(\mathbf{q}) = +1$ if $\mathbf{q} \in \mathcal{C}_{obs}$ and $c(\mathbf{q}) = -1$ if $\mathbf{q} \in \mathcal{C}_{free}$.

PD Formulation: We define global penetration depth as the minimum motion or transformation required to separate two intersecting objects A and B [Agarwal et al. 2000; Kim et al. 2002a]:

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

where $\mathbf{q}_0$ is an in-collision configuration and $\mathbf{q}$ is a configuration that lies in the contact space $\mathcal{C}_{cont}$. We use the notation $\mathrm{dist}(\cdot, \cdot)$ to represent the distance between two configurations, which may correspond to any metric defined on the $\mathcal{C}$-space. The point or configuration on the contact space for which $\mathrm{PD}(A, B)$ attains its minimal value is denoted as $\mathbf{q}_c = \mathrm{argmin}_{\mathbf{q} \in \mathcal{C}_{cont}} \mathrm{dist}(\mathbf{q}_0, \mathbf{q})$.

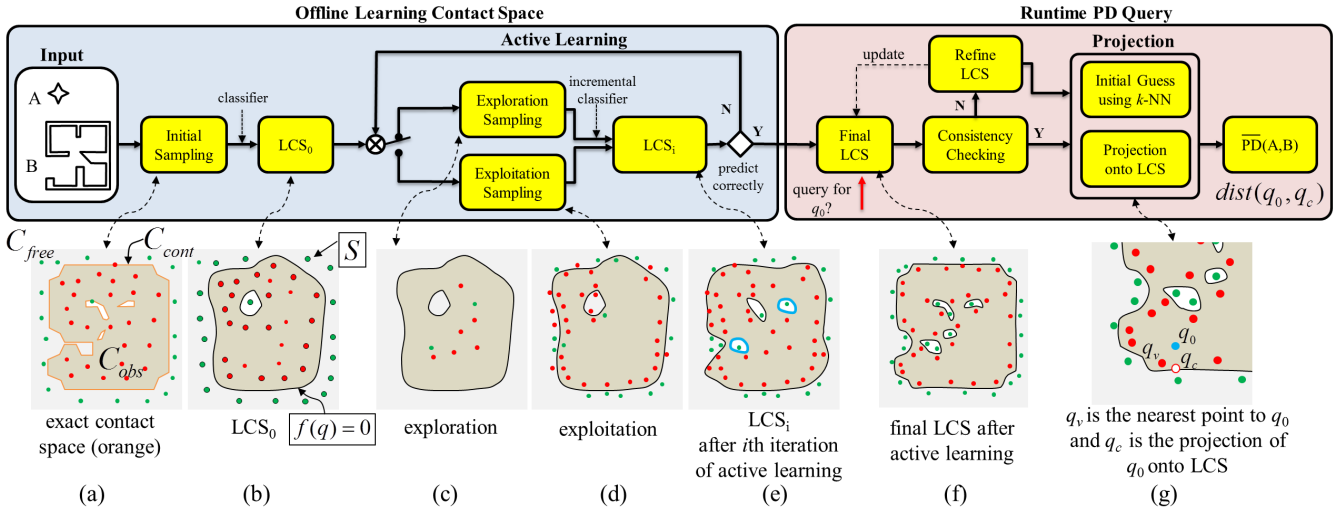


Figure 3: *This figure shows the offline computation pipeline and the runtime phase of our algorithm. The different approximations of LCS are shown below the corresponding stage. We use green points for collision-free configuration samples and red points for in-collision samples.*

In this paper, we mainly limit ourselves to translational and rotational motion and use appropriate $\mathrm{dist}(\cdot,\cdot)$ metrics. For translational PD (PD_t), $\mathrm{dist}(\cdot,\cdot)$ is the standard Euclidean distance metric between 3-DOF vectors corresponding to the configurations. Many distance metrics have been proposed for generalized PD (PD_g) computation, including weighted Euclidean distance [Wang et al. 2012b] or object norm [Je et al. 2012] or displacement distance metric [Zhang et al. 2007a]. We used the displacement distance metric in our algorithm, which is defined as:

$$\mathrm{dist}(\mathbf{q}_i,\mathbf{q}_j) = \mu_1 q_1^2 + \mu_2 q_2^2 + \mu_3 q_3^2 + q_4^2 + q_5^2 + q_6^2, \quad (2)$$

where (q_1,q_2,q_3) and (q_4,q_5,q_6) measure the relative rotation and the relative translation between two configurations $\mathbf{q}_i$ and $\mathbf{q}_j$, respectively. (q_0,q_1,q_2,q_3) corresponds to the relative quaternion between these two configurations, where $q_0 = (1-q_1^2-q_2^2-q_3^2)^{1/2}$. μ_i is the weight on the rotational component and is computed as [Zhang et al. 2007a]:

$$\mu_1 = \frac{4}{\mathrm{Vol}} I_{xx}, \quad \mu_2 = \frac{4}{\mathrm{Vol}} I_{yy}, \quad \mu_3 = \frac{4}{\mathrm{Vol}} I_{zz}, \quad (3)$$

where $\mathrm{diag}(I_{xx}, I_{yy}, I_{zz})$ represents the diagonal of the inertia matrix of object A and Vol is the volume of object A. For closed models, the volume Vol is computed based on a tetrahedral decomposition computed using an interior point. For other models, we use the volume of its bounding sphere. Note that a weighted metric is application dependent and we can adjust the relative weight of the translational and rotational component.

3.2 Active Machine Learning

Given the high combinatorial complexity of exact computation of contact space, our approach computes an approximation based on sampling and active machine learning. Active learning is a form of supervised machine learning. The key idea is that if a learning algorithm is allowed to choose the data from which it learns, that algorithm will perform better with less training. It is generally used in applications where labels are expensive to compute and has been widely used for web searching, email filtering, relevance feedback, computational biology [Murphy 2011] and text classification [Tong and Koller 2002]. In our application, the label of each sample is computed via exact collision detection, which is relatively expensive, especially for complicated meshes with many triangles. Therefore, active learning is a good choice to smartly refine the approximation of the contact space by choosing appropriate samples. Typically, the selections for active learning are made by a machine learning method which determines the informativity of the examples. In our case, we use support vector machines (SVM) which are used to analyze sample configurations and generate a decision function or *classifier* [Mohri et al. 2012]. The structure of SVMs can be exploited to determine which data points or samples should be chosen by the learning algorithm. A common strategy in active learning is to use a combination of exploration and exploitation [Huang et al. 2010]. Simply speaking, exploration is used to select samples in the undiscovered regions (i.e., far from the decision function), and exploitation is used to generate samples near the known regions (i.e., close to the decision function). In practice, the exploration-and-exploitation active learning algorithm is widely used in many applications, including computer vision, computational biology, machine learning, etc. Using active learning, the cost of sampling can be reduced, or, alternately, a more accurate approximation of contact space can be obtained for the same cost.

3.3 Approximate PD Formulation

Given the approximate representation of the contact space, we can then compute the approximate global PD by performing a nearest-neighbor query in the $\mathcal{C}$-space. Our formulation relies on generating a sufficient number of samples in the configuration space and using active learning and classification techniques to compute the approximation efficiently. We denote this approximation as *Learned Contact Space* (LCS). Based on LCS, we can, analogously, define an approximate penetration depth computation as

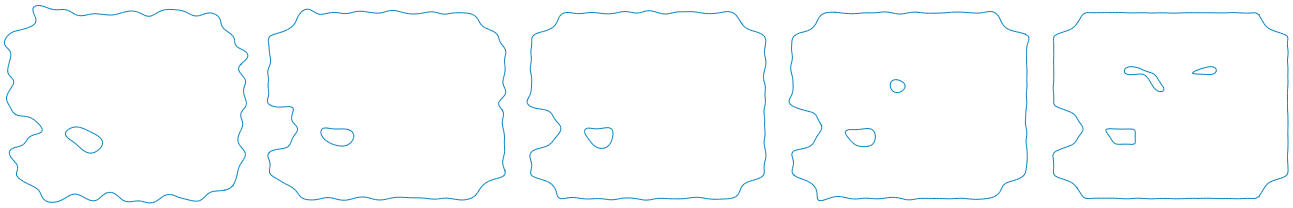
$$\overline{\mathrm{PD}}(A(\mathbf{q}_0), B) = \min_{\mathbf{q}\in LCS} \mathrm{dist}(\mathbf{q}_0,\mathbf{q}), \quad (4)$$

The accuracy of $\overline{\mathrm{PD}}$ is governed by the accuracy of LCS.

3.4 PD Computation

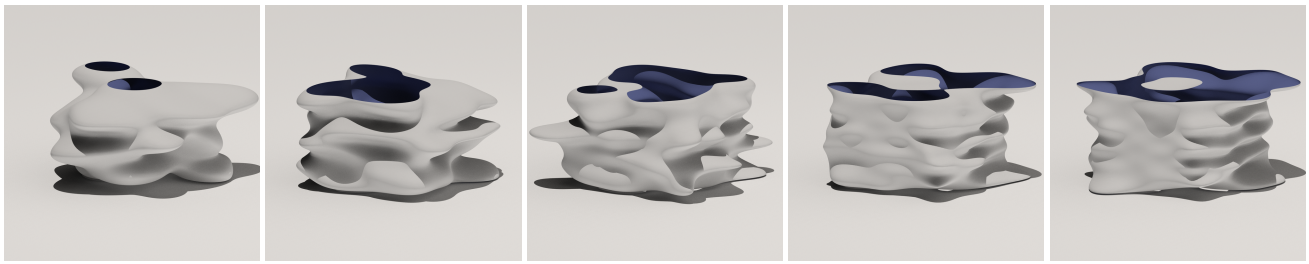
Our approach consists of two phases: offline learning of contact space (LCS) and runtime PD query. The overall pipeline of our algorithm is illustrated in Figure 3.

Offline Learning Phase: As shown in Figure 3-(a), we first generate a small set of uniform samples in a subspace of $\mathcal{C}$-space



(a) $LCS_0, |S| = 37$ (b) $LCS_4, |S| = 75$ (c) $LCS_8, |S| = 198$ (d) $LCS_{14}, |S| = 327$ (e) $LCS_{20}, |S| = 654$

Figure 4: *LCS computation using active learning for PD_t query between 2D star and room models shown as the input of learning pipeline in Figure 3. We highlight the number of support vectors corresponding to LCS_i. In our benchmarks, the algorithm can compute a good approximtion in a few iterations.*



(a) $LCS_0, |S| = 88$ (b) $LCS_2, |S| = 123$ (c) $LCS_5, |S| = 174$ (d) $LCS_9, |S| = 237$ (e) $LCS_{12}, |S| = 248$

Figure 5: *LCS computation using active learning for PD_g query between 2D non-convex shapes given in Figure 3. We show the approximation after i-th iteration and the number of support vectors. The vertical axis represents the rotational component of the $\mathcal{C}$-space.*

for two given objects. Next, we classify these configurations into $\mathcal{C}_{free}$ or $\mathcal{C}_{obs}$ by performing exact collision checking between the two objects using bounding volume hierarchies. Given the collision states (-1 or $+1$) of all configuration samples, a coarse approximation LCS_0 (Figure 3-(b)) is computed using classifiers. Next, we select new samples in $\mathcal{C}$-space to further improve the accuracy of the initial representation LCS_0 using active learning. We either select samples that are far away from prior samples (so-called *exploration*) (Figure 3-(c)), and near LCS_0 (so-called *exploitation*) (Figure 3-(d)). After the new samples are generated, we compute an updated approximation LCS_1 (Figure 3-(e)) based on incremental machine learning techniques. We repeat this process, generating a sequence of approximate representations LCS_0, LCS_1, ..., with increasing accuracy. This iterative process is repeated until the collision states of all the new samples can be correctly predicted by the current approximation. The final result LCS (Figure 3-(f)) corresponds to either a piecewise linear (for PD_t) or smooth (for PD_g) surface approximation of contact space.

Runtime Phase: We use the approximate contact space LCS, generated during the learning phase, to compute the penetration depth for a given in-collision query. As shown in Figure 3-(g), given a relative configuration $\mathbf{q}_0$, we perform a nearest-neighbor search to compute a configuration that is close to the decision function and project to the boundary of LCS. Let $\mathbf{q}_c$ correspond to the configuration on LCS that is closest to $\mathbf{q}_0$. Finally, the distance between $\mathbf{q}_0$ and $\mathbf{q}_c$ is computed using the appropriate distance metric $\mathrm{dist}(\cdot, \cdot)$ to approximate the PD.

4 Contact Space Learning

In this section, we present our algorithm for the offline learning of contact space and the computation of LCS. Different stages of this algorithms are shown in Figure 3.

4.1 Initial Sampling

We perform uniform sampling in $\mathcal{C}$-space to obtain a set of configuration points. Rather than sampling the entire $\mathcal{C}$-space, we generate samples in a subspace that contains $\mathcal{C}_{cont}$. Given two objects A and B, the contact space $\mathcal{C}_{cont}$ is contained in the in-collision space $\mathcal{C}_{obs}$ of their bounding volumes $BV(A)$ and $BV(B)$. In our case, we choose axis-aligned bounding boxes (AABB) as the underlying BVs for PD_t computation, due to their translational invariance in $\mathcal{R}^2$ and $\mathcal{R}^3$. Similarly, we use spheres as the underlying BVs for PD_g computation due to its translational and rotational invariance in $SE(2)$ and $SE(3)$.

4.2 Compute LCS_0

Given a set of k samples from $\mathcal{C}_{obs}(BV(A), BV(B))$, we perform exact collision queries between A and B to check whether these samples are in-collision space or not. Note that performing Boolean or discrete collision queries between complex models is a much easier problem as compared to PD computation, as shown in Section 6.4. Our goal is to learn an approximate representation, LCS_0 from these configurations. In particular, LCS_0 corresponds to a decision function $f(\mathbf{q}) = 0$ that is fully determined by a set of configurations S in $\mathcal{C}$-space. We refer to $f(\mathbf{q})$ as the *classifier* and use it to predict whether a given configuration $\mathbf{q}$ is collision-free ($f(\mathbf{q}) < 0$) or in-collision ($f(\mathbf{q}) > 0$). S corresponds to the *support vectors*, which is a small subset of configuration samples used in learning. Intuitively, S are the samples that are close to $\mathcal{C}_{cont}$.

There are possible methods that can be used to compute the approximate contact space. One possible alternative is to use surface fitting techniques to approximate the contact space using an implicit function, but it gets more challenging for high-dimensional spaces (e.g., 6-DOF $\mathcal{C}$-space). Another possibility is to use regression based learning techniques to approximate the contact space.

However, such techniques typically require an improved or continuous approximation of PD values at these samples, which is much harder to compute (as compared to discrete collision queries).

4.2.1 Nonlinear Classifier based on SVM

We use the classifier SVM [Vapnik 1995] to learn LCS_0 from the set of k configurations. A decision function generated by SVM is a smooth nonlinear surface. As the underlying samples can always be separated into collision-free and in-collision spaces, we use hard-margin SVM. Intuitively speaking, SVM maps the given samples $\{\mathbf{q}_i\}$ into a higher (possibly infinite) dimensional space by a function ϕ, which computes a mapping from an *input space* onto the *feature space*, which may be infinite dimensional. For LCS computation, the input space is the $\mathcal{C}$-space.

SVM computes a linear separating hyperplane in feature space characterized by parameters $\mathbf{w}$ and b, which corresponds to a nonlinear separating surface in the input space. In this case $\mathbf{w}$ is the normal vector to the hyperplane; and the parameter b determines the offset of the hyperplane from the origin along the normal vector. In the feature space, the distance between a hyperplane and the closest sample point is called the 'margin', and the optimal separating hyperplane should maximize this distance. The maximal margin can be achieved by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & c_i(\mathbf{w}\cdot\phi(\mathbf{q}_i)+b)\geq 1, \qquad 1\leq i\leq k. \end{aligned} \tag{5}$$

where $c_i \in \{-1,+1\}$ is the collision state of each sample $\mathbf{q}_i$..

Let $K(\mathbf{q}_i,\mathbf{q}_j)=\phi(\mathbf{q}_i)^T\phi(\mathbf{q}_j)$ represent the kernel function (i.e., a function used to calculate inner products in the feature space), the distance between two points $\phi(\mathbf{q}_i)$ and $\phi(\mathbf{q}_j)$ in the feature space can be computed as:

$$\begin{aligned} &\|\phi(\mathbf{q}_i)-\phi(\mathbf{q}_j)\| \\ &=\sqrt{K(\mathbf{q}_i,\mathbf{q}_i)+K(\mathbf{q}_j,\mathbf{q}_j)-2K(\mathbf{q}_i,\mathbf{q}_j)}. \end{aligned} \tag{6}$$

In our algorithm, we use the radial basis function (RBF) as the kernel: $K(\mathbf{q}_i,\mathbf{q}_j)=\exp(-\gamma\|\mathbf{q}_i-\mathbf{q}_j\|^2)$, where γ is a positive parameter and the corresponding ϕ function is infinite dimensional. In practice, we use $\gamma=20$ in our algorithm. We use RBF kernel because it keeps the distance ranking in both the input space and the feature space due to the fact that $\|\phi(\mathbf{q}_i)-\phi(\mathbf{q}_j)\|_2^2 = 2-2\cdot\exp(-\gamma\|\mathbf{q}_i-\mathbf{q}_j\|_2^2)$.

The solution of Equation 5 is a nonlinear surface in the input space (a hyperplane in the feature space) that separates collision-free and in-collision configurations. Based on the dual form of optimization in Equation 5, this solution can be formulated as:

$$f(\mathbf{q})=\mathbf{w}^*\cdot\phi(\mathbf{q})+b^*=\sum_{i=1}^{k}\alpha_i c_i K(\mathbf{q}_i,\mathbf{q})+b^*, \tag{7}$$

where $\mathbf{w}^*$ and b^* are the solutions of Equation 5 and $\alpha_i \geq 0$ correspond to each training sample's weight for the classification, which comes from the dual form of optimization in Equation 5. According to Karush-Kuhn-Tucker conditions, only a few of the α_i's are non-zero and the corresponding $\mathbf{q}_i$ are the *support vectors*, S. Intuitively, the support vectors are those samples very close to the separating hyperplane $f(\mathbf{q})=0$, as shown by the red and green configurations in Figures 3(b) and 3(g). Finally, LCS_0 consists of an implicit function $f_{LCS_0}(\mathbf{q})=f(\mathbf{q})$ and a set of samples $S_{LCS_0}=S$ (i.e., the support vectors), which are used to approximate the exact contact space.

4.3 Refine LCS_0 using Active Learning

We refine LCS_0 using active learning. The goal is to actively select new samples so that a better approximate contact space representation, LCS_1, can be obtained by incorporating these samples into LCS_0. We use a combination of exploration and exploitation [Huang et al. 2010]. The basic idea is to determine whether to explore or to exploit by flipping a biased coin with a probability (initially 0.5). If the result is a head, we apply exploration. If the result is a tail, we apply exploitation, and the given probability is adjusted according to the fraction of exploration samples whose collision states are correctly predicted by the current LCS_i. The new samples are used to update LCS_0 and generate a new approximation LCS_1 (or refine from LCS_i to LCS_{i+1}). We repeat the active learning step until all the new samples can be correctly predicted by the current LCS_i or the final result (represented as LCS) has sufficient accuracy to approximate $\mathcal{C}_{cont}$. Later, in Section 6 we show that active learning offers improved convergence as compared to uniform or random sampling schemes.

4.3.1 Exploration

It is possible that LCS_0 may miss some holes or components corresponding to collision-free regions, as compared with exact $\mathcal{C}_{cont}$, because no samples were generated inside those regions. As a result, there may be some portions that LCS_0 may incorrectly classify, as shown in Figure 3(c). In this case, exploration allows us to generate samples far away from prior samples in order to explore the regions not well-sampled by the current LCS_0. In our algorithm, we use random sampling to explore these new regions (Figure 3-(c)). As shown in Figure 3(e), two new collision-free regions (marked as blue curves) are found after exploration. After each exploration sampling step, we need to determine whether the exploration improves LCS_0 by computing the fraction of the new samples that are not correctly predicted by LCS_1. If this fraction is large (e.g., 0.3), then we increase the probability for exploration; otherwise we decrease it.

4.3.2 Exploitation

Exploitation allows us to generate samples near the decision function of a given approximation LCS_0. We use a simple method based on the *maximal margin* property of SVM. The maximal margin property [Vapnik 1995] states that the decision function will have the same distance to the support vectors with different labels (i.e., collision-free or in-collision). In order to obtain a sample near the decision function $f_{LSC_0}=0$, we first choose a pair of support vectors that are close to each other, but have opposite labels. Based on the maximal margin property, the midpoint of the two supporting vectors lies on or near the decision function. For nonlinear SVM,

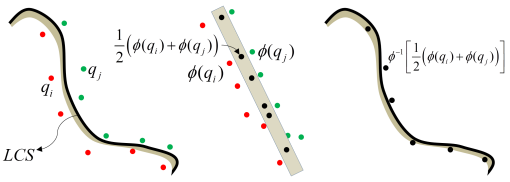


Figure 7: *Exploitation in SVM: (a) support vectors are on different side of the decision function ($\mathbf{q}_i$ and $\mathbf{q}_j$) in input space; (b) their midpoints (black points) are computed in the feature space; (c) the pre-image of the midpoints lies near the decision function and can be used for exploitation.*

the closest point and interpolation computations are performed in the feature space. As shown in Figure 7, we first use the distance

(a) $LCS_0, |S| = 231$ (b) $LCS_2, |S| = 385$ (c) $LCS_5, |S| = 869$ (d) $LCS_9, |S| = 1350$ (e) $LCS_{12}, |S| = 1572$

Figure 6: *LCS computation using active learning for PD_t query between 3D cup and spoon. We highlight the number of support vectors corresponding to LCS_i. In our benchmarks, the algorithm can compute a good approximation in a few iterations.*

metric mentioned in Equation 6 to find a pair of supporting vectors $\mathbf{q}_i$ and $\mathbf{q}_j$. Next, we compute their midpoint $\frac{1}{2}(\phi(\mathbf{q}_i) + \phi(\mathbf{q}_j))$ (shown as black points in Figure 7-(b)). However, the resulting midpoint may not have a pre-image in the input space. Therefore, we search the input space for a point $\mathbf{q}$ whose image $\phi(\mathbf{q})$ in feature space is closest to $\frac{1}{2}(\phi(\mathbf{q}_i) + \phi(\mathbf{q}_j))$:

$$\begin{aligned} &\min_{\mathbf{q}} \quad \|\frac{1}{2}(\phi(\mathbf{q}_i) + \phi(\mathbf{q}_j)) - \phi(\mathbf{q})\|_2 \\ \Leftrightarrow &\max_{\mathbf{q}} \quad K(\mathbf{q}, \mathbf{q}_i) + K(\mathbf{q}, \mathbf{q}_j). \end{aligned} \tag{8}$$

The solution is solved using an optimization solver based on sequential quadratic programming (SQP), where the midpoint in the input space $\frac{\mathbf{q}_i+\mathbf{q}_j}{2}$ is used as the initial guess. In our benchmarks, this optimization solver tends to converge quickly (less than 10 iterations).

4.4 Incremental Learning

Instead of computing a new decision function from scratch using all the previous samples, we apply incremental learning techniques to efficiently compute LCS_{i+1} from LCS_i. Intuitively, incremental learning utilizes a small set of new samples to update LCS_i. The decision function of LCS_i serves as the initial guess for generating a refined result LCS_{i+1}. Incremental SVM [Karasuyama and Takeuchi 2009] can be used to update the current result generated using SVM; the key is to retain the optimality condition of Equation 5 (i.e., the Kuhn-Tucker condition) on all prior samples while adding new samples. This is achieved by adjusting the coefficients α_i and b in Equation 7 and by migrating some samples in and out of support vector set S. The coefficient adjustment and the support vector migration are guided by the gradient of the objective function in Equation 7.

4.5 Terminating Active Learning

Active learning terminates when either of these conditions has been satisfied:

(a) The collision states of all the new samples generated during exploration and exploitation computation can be correctly predicted by the current approximation LCS_i.
(b) The total number of samples used in active learning iterations is more than a user given threshold.

The first condition guarantees that all the configurations used for learning LCS are consistent (i.e., they can be correctly predicted by LCS). This implies that the current LCS has a high probability of accurately approximating the underlying contact space. The second condition controls the error in PD computation. As more samples are used, we get a better approximation to $\mathcal{C}_{cont}$, and thereby lower PD error.

5 Runtime PD Queries

We use the approximate contact space, LCS, to perform PD queries at runtime. In this section, we give details on the runtime algorithm. It consists of two parts: local LCS refinement based on consistency checks and computing the nearest configuration on LCS.

5.1 Local LCS Refinement

Let $\mathbf{q}_0$ be a configuration that corresponds to overlapping rigid objects A and B. The exact collision check between these objects is performed using bounding volume hierarchies. We also compute the approximate collision state corresponding to $\mathbf{q}_0$ using LCS: i.e., check whether $(f(\mathbf{q}_0) > 0)$ as that corresponds to an in-collision configuration. It is possible that the collision state predicted using LCS may be different from that computed by the exact algorithm, which implies that LCS is not sufficiently accurate at approximating the contact space in the neighborhood of $\mathbf{q}_0$. In this case, we refer to $\mathbf{q}_0$ as an *inconsistent* configuration; otherwise, it is consistent. Generally, an inconsistent configuration occurs when the query is located in a $\mathcal{C}$-space region that is not well sampled during learning phase. During runtime phase, the inconsistent configuation rarely occurs, especially if the LCS is quite accurate. For example, for all the benchmarks we used in Table 1, the inconsistent configurations arise in 1% to 4% of all the query samples generated during runtime phase.

Our runtime algorithm first checks whether a given query $\mathbf{q}_0$ is consistent. If $\mathbf{q}_0$ is inconsistent, $\mathbf{q}_0$ corresponds to a collision-free configuration which is predicted in-collision by LCS $(f(\mathbf{q}_0) > 0)$ or an in-collision configuration which is predicted to be collision-free by LCS $(f(\mathbf{q}_0) < 0)$. We only consider the inconsistent configuration whose distance to LCS is more than a user-specified error threshold. In this case, we locally refine LCS by incorporating $\mathbf{q}_0$ into LCS using incremental learning (see Section 4.4). This local refinement of LCS improves the query efficiency and the accuracy of PD computation, as highlighted in Equation 11.

During each runtime query, we perform an incremental learning step for an inconsistent single configuration. As a result, its runtime overhead is $O(1)$. Moreover, this local refinement step improves the accuracy of LCS in regions, where more PD queries are performed by an application during runtime. As a result, this scheme exploits the spatial coherence in configuration space for nearby PD queries and results in more accurate answers for all those queries.

5.2 LCS Projection

Given a consistent configuration $\mathbf{q}_0$, we search for the closest configuration on LCS to compute the PD. In particular, we *project* $\mathbf{q}_0$ onto the decision boundary $f_{LCS} = 0$, and let $\mathbf{q}_c$ be the nearest configuration on LCS. In this case, the approximate PD is computed using $\mathrm{dist}(\mathbf{q}_0, \mathbf{q}_c)$ function. For SVM classifiers, the projection computation can be reduced to a constrained optimization problem:

$$\min_{\mathbf{q}} \quad \mathrm{dist}(\mathbf{q}_0, \mathbf{q}), \quad \text{subject to} \quad f_{LCS}(\mathbf{q}) = 0. \tag{9}$$

We use SNOPT package [Gill et al. 2005] to efficiently solve this optimization problem. A key challenge is to perform this projection efficiently, ensuring that the optimization algorithm is not trapped in a local minima, as the shape of the decision function can be rather complicated. In order to deal with these issues, we perform the computation in two phases: first, we perform a k-nearest-neighbor search in $\mathcal{C}$-space to compute the configuration $\mathbf{q}_v \in S_{LCS}$ (i.e., among the support vectors) that is closest to $\mathbf{q}_0$ based on our $\mathrm{dist}(\cdot,\cdot)$ metric. Next, we use $\mathbf{q}_v$ as an initial guess to the constrained optimization problem and compute the closest configuration on the LCS. Since $\mathbf{q}_v$ is a configuration very close to the decision boundary, it serves as a good initial guess.

Nearest Neighbor Search: We use different nearest-neighbor (NN) search algorithms to compute $\mathbf{q}_v$, depending on whether we are performing this search in 3-DOF $\mathcal{C}$-space or 6-DOF $\mathcal{C}$-space. For 3-DOF $\mathcal{C}$-space, $\mathrm{dist}(\cdot,\cdot)$ corresponds to the Euclidean distance metric, and we use a kd-tree to accelerate NN computation. For 6-DOF $\mathcal{C}$-space, $\mathrm{dist}(\cdot,\cdot)$ is the non-Euclidean displacement metric, as defined in Section 3. As a result, we use a hierarchical clustering algorithm for efficient NN search [Muja and Lowe 2009].

6 Analysis

In this section, we analyze various characteristics of our algorithm, including errors in PD computation, benefits of active learning, and time and space complexity.

6.1 Error in LCS and in PD Computation

Since our approach is probabilistic, we compute a bound on PD approximation based on *expected error* [Vapnik 1995], which corresponds to the average error when LCS is applied to predict the collision state or PD value for a new configuration in the $\mathcal{C}$-space. This error can be expressed as:

$$e_{\text{col}} = \mathbb{E}\left|e_{\text{cs}}(\mathbf{q})\right|, \tag{10}$$

where $e_{\text{cs}}(\mathbf{q}) = 0$ if $\mathbf{q}$ is a consistent configuration; otherwise $e_{\text{cs}}(\mathbf{q}) = 1$ if $\mathbf{q}$ is inconsistent. Expectation $\mathbb{E}$ is performed for a series of random configurations or queries (see appendix for more details). Typically, these queries arise from an application (e.g., dynamic simulation), and we assume that they follow a uniform distribution in $\mathcal{C}$-space.

The accuracy of approximate global PD computation is measured by the expected error that arises while using LCS to compute the PD for a random configuration in $\mathcal{C}$-space:

$$e_{\text{PD}} = \mathbb{E}\left|\overline{\text{PD}}(A(\mathbf{q}), B) - \text{PD}(A(\mathbf{q}), B)\right|. \tag{11}$$

Note that we scale the objects such that the maximum dimension of the subspace $\mathcal{C}_{obs}(BV(A), BV(B))$ is equal to 1. A small e_{col} usually implies a small value of e_{PD} and vice versa. There may be a few cases where e_{col} and e_{PD} can be quite different. For example, in a very narrow passage in $\mathcal{C}_{free}$, e_{PD} can be large but e_{col} is small. If $\mathcal{C}_{obs}$ is composed of many small isolated components, e_{col} can be large, whereas e_{PD} may be small.

6.2 Benefits of Active Learning

A key component of our algorithm is the computation of LCS by generating appropriate samples in the configuration space. The simplest choice is to perform uniform sampling in $\mathcal{C}_{obs}(BV(A), BV(B))$ or to use some other random sampling scheme. Instead, we use a combination of active and incremental learning techniques to refine LCS_i and improve its accuracy.

The time and space complexity of the LCS precomputation phase is a function of the number of samples used for active learning iterations. In particular, our goal is to ensure that the final LCS corresponds to a good approximation of $\mathcal{C}_{cont}$. Note that the number of samples required to achieve a given error bound e_{col} depends on both the active learning technique and the underlying classification method used within active learning iterations. In general, it is non-trivial to derive a tight bound on the number of samples for a specific combination of active learning and classification algorithms. However, we use some general results on sample complexity of active learning [Hanneke 2013] to show the benefits of our approach.

Theorem 1 *If the number of samples used in active learning iterations of LCS computation is more than N, where $N = \mathcal{O}(\log(1/(\epsilon\delta))$, then there exists one active learning technique which can guarantee that with probability at least $1-\delta$, the expected error of the LCS result will satisfy the bound $e_{col} \leq \epsilon$.*

Intuitively, this theorem says that there exists a special active learning technique that can be used to derive these bounds on LCS approximation. We give a proof of this theorem in the supplemental material based on CAL (Cohn-Atlas-Ladner) algorithm [Cohn et al. 1994]. This guarantees our LCS computation to have a bounded error with a high probability, if more than $N = \mathcal{O}(\log(1/(\epsilon\delta))$ samples are used. However, the CAL active learning algorithm is not practical [Hanneke 2013] and rather we use a combination of exploration-and-exploitation as the active learning (Section 4.3) in our LCS computation algorithm.

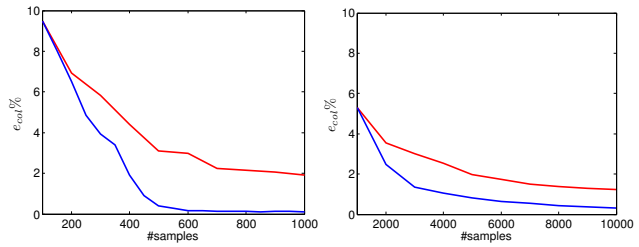
A lot of applications use exploration-and-exploitation active learning algorithm. In this case, we can expect that the use of exploration-and-exploitation could also result in a bound similar to Theorem 1, although the exact derivation of such a bound is a good topic for future research.

Since e_{col} and e_{PD} are closely related to each other, Theorem 1 also implies that e_{PD} decreases at an exponential rate with the number of samples. This is in contrast with using uniform sampling strategy to learn the contact space, in which LCS converges to the exact contact space at a polynomial rate when the number of samples increases [Mohri et al. 2012]:

Theorem 2 *When using uniform sampling, if the number of samples is more than N, where $N = \mathcal{O}(\frac{1}{2\epsilon^2}\log(2/\delta))$, then with probability $\geq 1-\delta$, we have the error bound $e_{col} \leq \epsilon$.*

Theorems 1 and 2 provide approximate bounds for the number of samples required to archive a specified error bound for the approximate LCS, when active sampling or uniform sampling is used, respectively. Since these bounds are based on unknown constants, we can't use them directly to compute the number of iterations in our contact space learning algorithm. At the same time, these two theorems provide theoretical justification to our use of active learning techniques to improve the accuracy and efficiency of our offline training algorithm. Since active sampling uses much fewer samples to achieve the same error bound than uniform sampling, active learning would outperform uniform sampling in spite of the computational overhead of exploitation and exploration operations.

We also measured the expected error, e_{col} or e_{PD}, in complex 2D



(a) e_{col} for 2D spiders (b) e_{col} for 3D cup-spoon

Figure 8: *Relative error convergence of active learning (blue) vs. uniform sampling (red) for 2D and 3D object pairs. These results demonstrate the benefits of active learning in terms of fewer samples and improved accuracy.*

and 3D benchmarks, as shown in Figure 8 (and also supplementary material). These figures verify the convergence of our contact space learning algorithm during offline learning phase, for both active learning and uniform sampling. It also demonstrates the high convergence rate and lower error in LCS and PD computation using active learning, with the same number of samples.

6.3 Benefits of Local Refinement

Our contact space and PD computation algorithms are probabilistic algorithms. Their accuracy is governed by the samples chosen during the learning phase, including initial samples and active learning, as well as the runtime queries. As more PD queries are performed within a subspace or a specific region of $\mathcal{C}$-space, the accuracy of LCS in that subspace or region tends to be higher. This is due to the local refinement step that is performed during runtime whenever we encounter an inconsistent query configuration. The incremental learning algorithm updates LCS around the query configuration by taking into account local information in $\mathcal{C}$-space. In many applications, including dynamic simulation, haptics, or motion planning, a high proportion of sample queries correspond to nearby positions of the two objects A and B. As a result, the runtime query configurations are relatively close to each other in $\mathcal{C}$-space and the local refinement step improves the accuracy of LCS in that region. This implies that as more queries are performed in a localized region of $\mathcal{C}$-space, the accuracy of LCS and PD queries improves. Our algorithm does not make any assumptions about the application or the distribution of runtime query configurations. We expect that the accuracy of local refinement will improve at the rate given by uniform sampling (i.e., Theorem 2), rather than the exponential rate of active learning. In other words, after generating $N = \mathcal{O}(1/\epsilon^2)$ samples within a subspace at runtime, the expected error locally around those samples will be less than ϵ.

6.4 Time and Space Complexity

The precomputation or learning phase is performed for each object pair (A, B) in the environment. The exact collision check is performed using precomputed bounding volume hierarchies. Given two objects with m and n triangles, the expected cost of a single exact collision query is $T_{col} = O(\log m + \log n)$.

Offline Learning: The timing complexity for the learning phase can be estimated as

$$(T_{LCS_0} + \sum_{i=1}^{I_{AL}} (T_{ES_i} + T_{LCS_i})) + T_{col} \cdot \sum_{i=1}^{I_{AL}} N_{LCS_i}, \qquad (12)$$

where T_{LCS_0} is the time complexity to learn the initial approximation; T_{ES_i} is the time cost to perform exploitation sampling or exploration sampling in the i-th iteration of active learning; T_{LCS_i} is the time cost for the i-th step incremental learning; I_{AL} is the number of iterations performed during active learning. We also denote the number of new samples generated during LCS_i as N_{LCS_i}. We need to perform collision checking for each sample generated during the learning phase; hence the collision cost is $T_{col} \cdot \sum_i N_{LCS_i}$.

T_{LCS_0} complexity is governed by the SVM classifier. SVM computation boils down to solving a constrained quadratic optimization problem using the interior point or conjugate gradient method and its worst case complexity is about $\mathcal{O}(N_{LCS_0}^{2.3})$ [Bottou and Lin 2007].

Incremental learning combines each new sample into LCS within constant time, and hence we have $T_{LCS_i} = \mathcal{O}(N_{LCS_i})$. T_{ES_i} is the time cost for exploitation sampling or exploration sampling. For exploration, $T_{ES_i} = \mathcal{O}(N_{LCS_i})$. The time complexity for exploitation sampling is $\mathcal{O}(|S_{LCS_i}|)$ as we perform interpolation between each support vector of LCS_i and its k-nearest neighbors, which can be bounded above as $\mathcal{O}(\sum_i N_{LCS_i})$.

Overall, the timing complexity for the learning phase is $\mathcal{O}(\log(\frac{1}{\epsilon}) \sum_i N_{LCS_i} + N_{LCS_0}^{2.3}) + T_{col} \cdot \sum_i N_{LCS_i}$. The space complexity of our algorithm is linear in the number of samples used during the learning and runtime phases and linear in the size of support vectors in the final LCS representation.

Runtime Query: The timing complexity in the runtime query phase depends on $|S_{LCS}|$, i.e., the number of support vectors in LCS. $|S_{LCS}|$ is related to the smoothness of exact $\mathcal{C}_{cont}$, and not so much on the geometric complexity of A and B (see Figures 4, 5, 6 and 10). For example, the $\mathcal{C}_{cont}$ of a sphere and another object (i.e., the offset surface) is always smooth, and therefore a small S_{LCS} is sufficient to generate a good approximation of $\mathcal{C}_{cont}$. We also notice that in our benchmarks, where $|S|$ for the teeth model (40K triangles) is comparable or higher than bunnies (70K triangles), dragon (230K triangles), and Buddha (1M triangles) models. Furthermore, we generated different low-polygon count representations of the Buddha models and observed similar performance on all these approximations (see supplementary material). In other words, the size of S_{LCS} depends on the combinatorial complexity of C_{cont}. Moreover, the size of S_{LCS} can be used to control the tradeoff between the accuracy of PD computation and the query efficiency.

7 Implementation and Performance

In this section, we highlight the performance of our algorithm on complex benchmarks and compare it with prior techniques. We implemented our algorithm using C++ under Visual Studio 2010 and Windows 7. The two main routines needed during the learning phase are exact collision checking between polygonal models and computing the approximation using support vector machines. At runtime, we need the capability to perform a nearest-neighbor query in the configuration space and to compute a projection using constrained optimization. We used the OBBTree algorithm [Gottschalk et al. 1996] for exact collision detection between polygonal objects. We also used a variant of GJK algorithm [van den Bergen 2001] to compute translational penetration depth between convex polytopes to compare the performance with prior methods. In our implementation, we have set $\epsilon = 2.5\%$ and $\delta = 0.01$.

7.1 Benchmarks

We have used many complex benchmarks to evaluate the performance of our algorithm. They are shown in Figure 1 and Figure 2

(also refer to the accompany video). In the simulation, there are multiple contacts between the overlapping objects and we compute PD_t and PD_g between them. The performance of the learning and runtime phases are shown in Table 1.

We precompute BVHs for collision detection, which has a linear memory complexity for each object. For each different type of object pair, we precompute their LCS, which takes about 5KB (star-box)-110KB (teeth, dragon, bunny, Buddha) memory.

7.2 Physically Based Simulation using PD

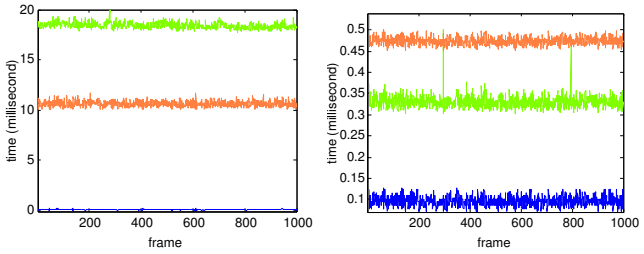
Penetration depth has been used in many dynamic simulators to compute collision response based on penalty forces or constraint-based solvers. We have integrated our new PD algorithm into the well-known game physics engines, Box2D [Catto 2010] and Bullet [Coumans 2010]. These engines have support for PD computation based on convex decomposition and computing the local translational penetration depth between convex polytopes [van den Bergen 2001]. However, convex decomposition can result in a high number of convex pieces, Moreover, the decomposition-based approach is mainly limited to closed objects and doesn't guarantee that two overlapping non-convex objects can separate as they only compute local PD using the convex pairs.

Contact Points and Normals: For an inter-penetration configuration $\mathbf{q}_0$ and its resulting contact configuration $\mathbf{q}_c$, the contact points and contact normal can be computed in the workspace for two objects. First, for the contact configuration $\mathbf{q}_c$, its nearest collision-free configuration can be computed using support vectors based on k-nearest-neighbor search in $\mathcal{C}$-space. Next, a set of contact features (including contact points, edges, or faces) and contact normals of the given two objects can be computed using the proximity query algorithm [Larsen et al. 2000]. The performance of contact computation is similar to collision queries (tens of milliseconds) and is a fraction of PD computation. For collision response, we treat each contact point as a planar constraint and also cluster the nearby contacts. A constraint force proportional to the penetration depth is applied to resist, and ultimately eliminate the penetration. The PD can be also used to estimate the time of contact to apply impulsive forces in impulse-based collision response methods. Reliable multiple contact points can be obtained using perturbation and persistent contact caching techniques [Coumans 2010]. Our method can efficiently compute the contact features and normals for non-convex objects, while previous methods like [Coumans 2010] only handle convex objects.

Box2D uses PD computation in impulse-based collision response algorithm. We demonstrate the performance of our algorithm on two complex benchmarks (Figure 1): (1) angry bird characters falling into a complex chute; (2) Nazca spiders rolling in a tumbler. We precompute the LCS approximation in 3-DOF $\mathcal{C}$-space. The convex decomposition results in 17, 30, and 32 convex pieces for BigRedBird, WhiteBird and GreenPig models, respectively. Moreover, the Nazca spiders decomposed into 77 convex pieces. We have observe nearly 20 times improvement in PD query using our active learning algorithm over techniques based on convex decomposition used in Box2D (see Figure 9-(a)(b)). The collision response algorithm is based on Box2D implementation.

Bullet uses PD computation to handle penetrations in their constraint-based solver. We demonstrate the benefits of our PD computation algorithm in three scenarios (shown in Figure 1 or the video): (1) interlocking 10 rings; (2) a rainfall of $1,000$ rings; (3) collapse of a tower composed of $5,500$ rings. Each ring consists of 256 triangles. We precompute the LCS approximation in 6-DOF $\mathcal{C}$-space and use it to perform PD queries during the simulation. Each ring is decomposed into 16 convex pieces. As compared to the convex decomposition based algorithm used in Bullet, our PD computation algorithm is about an order of magnitude faster. We use the standard implementation of contact normal and collision response forces computation available in Bullet.

Complex 3D Models: We have evaluated the performance of algorithm on many complex models corresponding to cup-spoon, moving teeth, bunnies, dragons and Buddha models (see Figure 1). The exact motion trajectories are shown in the video and we performed LCS computation in 6D space. We observe more than an order of magnitude performance improvement than prior methods (see Figures 9, 11, and supplementary material).



(a) PD_g: cup-spoon (b) PD_g: rings (Bullet)

Figure 9: *Relative performance of PD computation for different benchmarks: The blue curve represents the query time computed by our approximate PD_g algorithm. The green curve corresponds to the query time computed using convex decomposition and local PD between convex pairs. The orange curve represents the PD_g query time computed using point-based approximation [Lien 2009].*

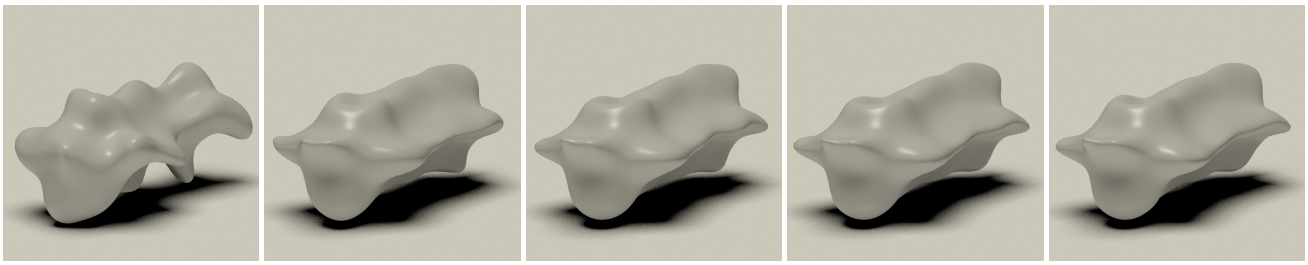
7.3 Comparison with Prior Methods

Most practical algorithms perform local analysis of the intersection regions and compute local PD. Other techniques use distance fields and can be accelerated using GPUs. In practice, these techniques are quite fast and can also handle deformable models. On the other hand, our global PD algorithm involves preprocessing and is mainly designed for rigid objects. The performance of our runtime query (e.g., about $0.1 \sim 2$ millisecond) is comparable or faster than these local PD computation algorithms. The main benefit of our approach over local PD methods is computation of global translational and rotational PD, which provides a more reliable measure of separating two overlapping objects. Other algorithms reduce PD computation to constrained optimization [Nawratil et al. 2009; Zhang et al. 2007a; Je et al. 2012]. In these techniques, a sequence of configuration samples on the contact space are iteratively computed until a local minimum configuration is found. The performance of these algorithms heavily relies on the initial guess of the configuration, and it is hard to provide error bounds in terms of global PD (see Figure 11). Basically, they are useful for computing locally optimal PD between non-convex objects. The approximate PD computed by our global algorithm can be used as an initial guess for these optimization-based techniques and thereby improve their accuracy.

In order to evaluate the error in our approximate PD computation algorithm, we need to compute very accurate (almost exact PD) between the two objects. For translational PD, the exact PD can be obtained by computing the Minkowski sum between two objects. However, for complex 3D objects like teeth or dragon, it is very hard to compute exact Minkowski sum due to its combinatorial complexity. Instead we use the point-based algorithm [Lien 2009] to approximate the PD and estimate the error of our algorithm. For generalized PD, the exact PD computation is even harder. Therefore, we approximate the exact contact space by many slices of Minkowski sums. Intuitively speaking, we sample many rotations in rotation space and then compute Minkowski sums for all

Model		Offline Learning LCS								Runtime Query				
		Initial Learning		Active Learning				total (s)	mem	time (ms)				$e_{\text{PD}}(\%)$
		#smpls	time (s)	#smpls	$\|S\|$	e_{col} (%)	time (s)			NN	projection	refine	total	
2D PD_t	star vs. room	100	0.006	1000	374	1.88	0.15	0.156	4.4	0.065	0.02	0.03	0.115	0.023
	monkeys	100	0.4	1000	346	0.11	2.74	3.14	4.2	0.06	0.01	0.03	0.10	0.008
	spiders	100	0.01	1000	389	1.37	0.27	0.28	4.7	0.066	0.01	0.02	0.096	0.025
3D PD_t	star vs. spoon	1000	0.08	10000	1105	0.59	1.245	1.33	17	0.43	0.21	0.02	0.66	0.012
	cup vs. spoon	1000	0.25	10000	1472	0.75	4.46	4.81	23	0.54	0.22	0.03	0.79	0.019
	rings	1000	0.20	10000	1224	0.56	11.99	12.01	19	0.66	0.12	0.05	0.83	0.016
	teeth	1000	0.33	10000	2132	1.3	43.21	43.54	34	1.3	0.2	0.08	1.58	N/A
	bunnies	1000	0.15	10000	666	1.7	36.49	36.64	11	0.1	0.12	0.04	0.26	2.0
	dragons	1000	0.17	10000	854	1.8	31.11	31.28	14	0.13	0.11	0.05	0.29	1.9
	Buddha	1000	1.7	10000	1384	1.8	37	38	22	0.18	0.10	0.09	0.37	1.8
2D PD_g	star vs. room	100	0.005	2000	436	2.0	1.276	1.281	6.9	0.08	0.03	0.02	0.13	0.021
	monkeys	100	0.42	2000	545	0.43	5.84	6.26	8.7	0.07	0.02	0.02	0.11	0.013
	spiders	100	0.011	2000	540	0.8	1.16	1.17	8.6	0.08	0.02	0.01	0.11	0.018
3D PD_g	star vs. spoon	1000	0.095	10000	1731	1.9	37.49	37.58	48	0.5	0.25	0.05	0.80	N/A
	cup vs. spoon	1000	0.3	10000	2107	1.2	78.34	78.64	59	0.3	1.0	0.03	1.33	N/A
	rings	1000	0.25	10000	1977	1.3	223.1	223.4	55	0.82	0.21	0.03	1.06	N/A
	teeth	1000	0.54	10000	3216	2.8	476.43	476.97	90	2.2	0.2	0.04	2.44	N/A
	bunnies	1000	0.33	10000	2283	3.1	342.31	342.64	64	0.89	0.12	0.02	1.03	N/A
	dragons	1000	0.37	10000	2387	2.8	378.92	477.29	69	1.01	0.18	0.03	1.22	N/A
	Buddha	1000	2.3	10000	3765	2.7	643	645	105	1.20	0.28	0.07	1.55	N/A

Table 1: *Performance of our PD algorithm on 2D and 3D models: The learning phase includes the number of samples, size of support vectors, final memory (KB) usage and precomputation time. We also give a timing breakdown of runtime query. The PD error is computed by comparing the accuracy of $\overline{PD}$ with prior algorithms used for PD computations. For accurate PD_t computation, we use the accurate, offline algorithm of [Lien 2009] or using a combination of convex decomposition and Minkowski sums. Since no accurate and efficient algorithms are known for many PD queries (e.g., PD_t computation for non-closed meshes like teeth model; PD_g computation for 3D models), we don't analyze the accuracy of our algorithm in such cases (shown as N/A).*



(a) $LCS_0, |S| = 115$ (b) $LCS_4, |S| = 608$ (c) $LCS_8, |S| = 1081$ (d) $LCS_{12}, |S| = 1407$ (e) $LCS_{15}, |S| = 1650$

Figure 10: *LCS computation using active learning for PD_t query between 3D star and spoon models. We highlight the number of support vectors corresponding to LCS_i. In our benchmarks, the algorithm can compute a good approximtion in a few iterations.*

the rotations. The combination of these Minkowski sums is used as an approximation of the contact space. We label the accurate PD computed using these offline techniques as "*nearly exact PD*" for our algorithm and comparisons. In practice, our approach is more than an order of magnitude faster than other algorithms that are based on convex decomposition (e.g., [Kim et al. 2002b] for PD_t; [Zhang et al. 2007b] for PD_g) or point-based approximations. We have compared the runtime performance of our algorithm with these prior global methods in Figure 9. More benchmarks, results and comparison are given in the supplementary material.

Figure 11 show the timings of PD_t runtime query using our algorithm against the most recent work, PolyDepth [Je et al. 2012]. Each benchmark corresponds to a pre-defined trajectory of configurations. We also use Lien's offline method [Lien 2009] to compute the Minkowski sum and compute the translational PD for each configuration in the pre-defined trajectory. The results are used as the ground-truth for translational PD. Next, we compute PD_t using our method as well as PolyDepth. From the comparison results shown in Figure 11, we observe that our method is faster and more accurate than PolyDepth (especially for deep penetrations), as PolyDepth uses a local optimization scheme which tends to be sensitive to the choice of initial guess. The ground truth of Lien's offline method is more accurate or smoother than our online method. However, it is more time consuming and is limited to translational PD computation.

8 Limitations and Conclusions

We have presented a novel approach to the computation of translational and generalized PD between polygonal models. The main idea is to sample the configuration space and approximate the contact space based on machine learning classifiers. We use support vector machines to approximate the contact space, and the runtime PD query is reduced to nearest neighbor computation. Furthermore, we use active learning techniques to select the samples during precomputation. The overall approach is general and applicable to all polygonal models. We have demonstrated the interactive performance of our algorithm on complex, non-convex models and have also used them for collision response in game physics engines. To

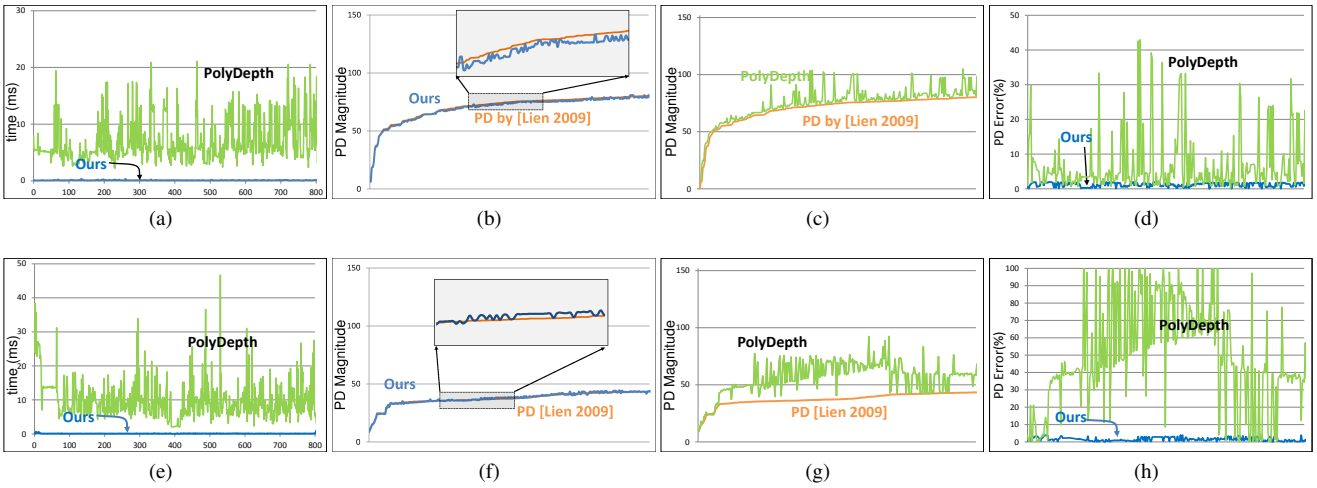


Figure 11: *The performance and accuracy comparison with PolyDepth [Je et al. 2012] on bunny-bunny (first row, from (a) to (d)) and dragon-dragon (second row, from (e) to (h)) benchmarks. (a) computational time (on average, the time for bunny-bunny benchmark is 0.10ms based on our algorithm vs. 7.15ms in PolyDepth; the time for dragon-dragon benchmark is 0.12ms based on our algorithm vs. 9.86ms in PolyDepth); (b) accuracy comparison between our interactive algorithm vs. offline algorithm based on Minkowski sum [Lien 2009]; (c) accuracy comparison of PD computation between PolyDepth vs. offline algorithm based on Minkowski sum [Lien 2009]; (d) our global PD algorithm (blue) has lower error as compared to PolyDepth that performs local optimization.*

the best of our knowledge, this is the first approach that is able to compute global and reliable PD between rigid models at interactive rates.

Our approach has a few limitations. The precomputation phase is performed for every object pair in the simulation. In the worst case, its complexity can grow as a quadratic function of the number of objects in the simulation. The accuracy and running time of our learning phase is a function of the combinatorial complexity of the contact space and the sampling scheme. It is possible that our method may not generate sufficient number of samples in small, isolated components of contact space, or may take a high number of iterations. The overall approach is probabilistic, and all our error bounds are derived in terms of expected error. Many times the solution to the penetration depth problem (Section 3.1) is not unique or differentiable. Since we compute a bounded-error approximation of PD, there could be multiple solutions that satisfy those error bounds. This discontinuity in PD formulation and computation can cause instability in collision response for haptic rendering, and our approach can be extended to compute continuous penetration depth [Zhang et al. 2013]. In complex rigid-body simulations with multiple objects, global PD computation can improve the accuracy of the simulation, but cannot guarantee that it is totally collision-free.

There are many avenues for future work, including overcoming the stated limitations. The basic components of our learning and run-time phases, such as SVM learning, collision detection, and nearest-neighbor computation, can be accelerated using GPU parallelism. We can use other active learning techniques to improve the sampling as well as other classifiers or learning techniques to improve the accuracy or convergence of LCS. It would be useful to derive tight theoretical error bounds (e.g., Theorem 1) for active learning algorithms based on exploitation-and-exploration. It would also be useful to extend the approach to articulated models that takes into account self-collisions between various links. In order to handle deformable models, we would like to develop incremental techniques that can refine the contact space approximation for deforming objects. It would be useful to this approach for other PD formulations, such as penetration volume [Weller and Zachmann 2009], which can result in continuous response forces. We also need improved algorithms for collision response that can guarantee collision-free simulations for interactive applications.

Acknowledgements

This research is supported in part by ARO Contract W911NF-10-1-0506, NSF awards 0917040, 0904990, 1000579 and 1117127 and Intel. We are also grateful to the reviewers for their feedback.

References

AGARWAL, P. K., GUIBAS, L. J., HAR-PELED, S., RABINOVITCH, A., AND SHARIR, M. 2000. Computing the penetration depth of two convex polytopes in 3D. In *Proceedings of Scandinavian Workshop on Algorithm Theory*, 328–338.

BARAFF, D., AND WITKIN, A. 2001. *Physically Based Modeling*. ACM SIGGRAPH Course Notes.

BOTTOU, L., AND LIN, C.-J. 2007. Support vector machine solvers. In *Large Scale Kernel Machines*, L. Bottou, O. Chapelle, D. decoste, and J. Weston, Eds. MIT Press, 301–320.

CATTO, E., 2010. Box2D: A 2D physics engine for games. `http://box2d.org`.

COHN, D., ATLAS, L., AND LADNER, R. 1994. Improving generalization with active learning. *Machine Learning 15*, 2, 201–221.

COUMANS, E., 2010. Bullet physics library. `http://bulletphysics.org`.

DOSHI, F., BRUNSKILL, E., SHKOLNIK, A., KOLLAR, T., AND ROHANIMANESH, K. 2007. Collision detection in legged locomotion using supervised learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

GILL, P. E., MURRAY, W., AND SAUNDERS, M. A. 2005. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review 47*, 1, 99–131.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of SIGGRAPH*, 171–180.

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

HANNEKE, S. 2013. A statistical theory of active learning. *Foundations and Trends in Machine Learning*, 1–212.

HEIDELBERGER, B., TESCHNER, M., KEISER, R., MLLER, M., AND GROSS, M. H. 2004. Consistent penetration depth estimation for deformable collision response. In *International Fall Workshop on vision, modeling and visualization*, 339–346.

HUANG, S.-J., JIN, R., AND ZHOU, Z.-H. 2010. Active learning by querying informative and representive examples. In *Proceedings of Advances in Neural Information Processing Systems*.

JE, C., TANG, M., LEE, Y., LEE, M., AND KIM, Y. J. 2012. PolyDepth: Real-time penetration depth computation using iterative contact-space projection. *ACM Trans. Graph. 31*, 1 (Feb.), 5:1–5:14.

KARASUYAMA, M., AND TAKEUCHI, I. 2009. Multiple incremental decremental learning of support vector machines. In *Proceedings of Advances in Neural Information Processing Systems*.

KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2002. DEEP: Dual-space expansion for estimating penetration depth between convex polytopes. In *Proceedings of International Conference on Robotics and Automation*, 921–926.

KIM, Y. J., OTADUY, M. A., LIN, M. C., AND MANOCHA, D. 2002. Fast penetration depth computation for physically-based animation. In *Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation*, 23–31.

LARSEN, E., GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 2000. Fast proximity queries with swept sphere volumes. In *International Conference on Robotics and Automation*, 3719–3726.

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

LIEN, J.-M. 2009. A simple method for computing minkowski sum boundary in 3D using collision detection. In *Algorithmic Foundation of Robotics VIII*, vol. 57 of *Springer Tracts in Advanced Robotics*. Springer Berlin / Heidelberg, 401–415.

MOHRI, M., ROSTAMIZADEH, A., AND TALWALKAR, A. 2012. *Foundations of Machine Learning*. The MIT Press.

MUJA, M., AND LOWE, D. G. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application*, 331–340.

MURPHY, R. F. 2011. An active role for machine learning in drug development. *Nature Chemical Biology 7*, 6, 327–330.

NAWRATIL, G., POTTMANN, H., AND RAVANI, B. 2009. Generalized penetration depth computation based on kinematical geometry. *Computer Aided Geometric Design 26*, 4 (May), 425–443.

PAN, J., CHITTA, S., AND MANOCHA, D. 2011. Probabilistic collision detection between noisy point clouds using robust classification. In *Proceedings of International Symposium on Robotics Research*.

REDON, S., AND LIN, M. C. 2006. A fast method for local penetration depth computation. *Graphical Tools 8*, 1, 63–70.

TANG, M., LEE, M., AND KIM, Y. J. 2009. Interactive Hausdorff distance computation for general polygonal models. *ACM Trans. Graph. 28*, 3, 74:1–74:9.

TANG, M., MANOCHA, D., OTADUY, M. A., AND TONG, R. 2012. Continuous penalty forces. *ACM Trans. Graph. 31*, 4, 107:1–107:9.

TONG, S., AND KOLLER, D. 2002. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Res. 2*, 45–66.

VAN DEN BERGEN, G. 2001. Proximity queries and penetration depth computation on 3D game objects. In *Game Developers Conference*.

VAPNIK, V. N. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

WANG, B., FAURE, F., AND PAI, D. K. 2012. Adaptive image-based intersection volume. In *Proceedings of SIGGRAPH*, 97:1–97:9.

WANG, D., LIU, S., ZHANG, X., AND XIAO, J. 2012. Configuration-based optimization for six degree-of-freedom haptic rendering for fine manipulation. *IEEE Transactions on Haptics 5*, 4, 332–343.

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

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

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

ZHANG, X., KIM, Y. J., AND MANOCHA, D. 2013. Continuous penetration depth. *Computer-Aided Design*. to appear.