Efficient Penetration Depth Approximation using Active Learning

Supplementary Material (paper # 0122)

1 Background

In this section, we give background related to some concepts and terminology used in the paper.

1.1 Configuration Space and Contact Space

Given two objects A and B, we denote their configuration space as $\mathcal{C}$-space and each configuration or point in $\mathcal{C}$-space corresponds to relative configuration (i.e., position and orientation) of A w.r.t 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 into account 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* 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 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 1 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 $\mathbf{q} \in \mathcal{C}_{free}$.

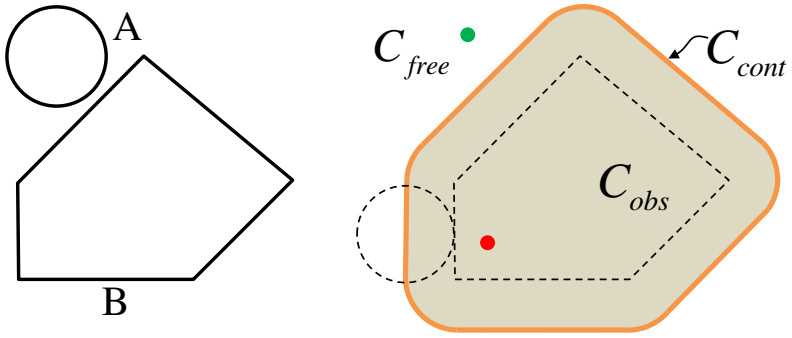


Figure 1: *The contact space of two objects. The orange curve highlights the contact space $\mathcal{C}_{cont}$ of A and B. A point inside/on the orange curve belongs to $\mathcal{C}_{obs}$ and a point outside the orange curve belongs to $\mathcal{C}_{free}$. The red and green points will be used to denote configurations in $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$, respectively. Intuitively, $\mathcal{C}_{cont}$ is the boundary that separates in-collision and collision-free configurations.*

1.2 Expected Error

As our PD computation method is a probabilistic approach, we use *expected* error to evaluate the accuracy of our method. Expected error is widely used in supervised and machine learning [Vapnik 1995]. Given a set of training samples (i.e., training set) $\{(\mathbf{q}_1, c_1), ..., (\mathbf{q}_N, c_N)\}$, where $\mathbf{q}_i$ is a sample, c_i is its label for collision state and N is the total number of training samples, our learning algorithm computes a function $f : \mathcal{C}\text{-space} \rightarrow \{-1, +1\}$. The expected error of $f(\cdot)$ on a new sample in the $\mathcal{C}$-space is computed as $\mathbb{E}(f) = \int_{\mathbf{q} \in \mathcal{C}\text{-space}} p(\mathbf{q}) L(f(\mathbf{q}), c)$, where $p(\mathbf{q})$ is the probability that $\mathbf{q}$ will appear as a query, c is the actual collision state of $\mathbf{q}$, and $f(\mathbf{q})$ is the collision state predicted by LCS. $L(\cdot)$ is the loss function (e.g., $\| \cdot \|_1$) used to measure the error between $f(\mathbf{q})$ and c. $L(f(\mathbf{q}), c)$ corresponds to $e_{\text{col}}(\mathbf{q})$, the average error when LCS is applied to predict the collision state (also see Equation 8 in the paper).

$$e_{\text{col}} = \mathbb{E} \left| e_{\text{cs}}(\mathbf{q}) \right|, \tag{1}$$

where $e_{\text{cs}}(\mathbf{q}) = 0$ if $\mathbf{q}$ is consistent configuration; otherwise $e_{\text{cs}}(\mathbf{q}) = 1$ if $\mathbf{q}$ is inconsistent.

We assume $p(\mathbf{q})$ to be uniformly distributed. In many simulation applications, some configurations may appear more frequently than others, and therefore $p(\mathbf{q})$ may be non-uniform. In these cases, exact computation of $\mathbb{E}(f)$ can be non-trivial. Therefore, we use random $M (\ll N)$ samples within $\mathcal{C}$-space and estimate $\mathbb{E}(f)$ as $\hat{\mathbb{E}}(f) = \sum_{j=N+1}^{M+N} L(f(\mathbf{q}_j) - c_j)$. We use these error bounds as e_{col} in Table 1 in the paper.

Based on a learned LCS, we can analogously compute another function $\mathcal{C}\text{-space} \rightarrow \mathbb{R}$ to estimate the PD value of a given query. The error can be formulated similarly as follows (also see Equation 11 in the paper).

$$e_{\text{PD}} = \mathbb{E} \left| \overline{\text{PD}}(A(\mathbf{q}), B) - \text{PD}(A(\mathbf{q}), B) \right|. \tag{2}$$

2 Proof and Experimental Errors

2.1 Theorem 1 and its Proof

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 $e_{col} \leq \epsilon$.*

To prove Theorem 1, we need to show that our LCS computation pipeline can find a special active learning algorithm (which can be some algorithm other than the exploration-and-exploitation algorithm) with the non-linear SVM as the classification algorithm during each iteration. In this case, we can guarantee that with probability at least $1-\delta$, the expected error of the LCS result will satisfy the bound $e_{\text{col}} \leq \epsilon$, if more than $N = \mathcal{O}(\log(1/(\epsilon\delta))$ samples are used.

The special active learning algorithm used in the proof is CAL (Cohn, Atlas, and Ladner [Cohn et al. 1994]) active learning technique [Hanneke 2013]. CAL is one of the earliest and most elegant general-purpose active learning algorithms purposed by Cohn, Atlas, and Ladner [Cohn et al. 1994]. It is one example of the disagreement-based active learning and is based on a very simple principle: never request the label of a data item if we can derive it from information already available. In some sense, it represents the least we could expect from a reasonable active learning algorithm. This technique does not always yield optimal number of data items required for a given error bound. However, due to its elegance and other favorable properties, it is one of the most commonly-studied technique in theory and it is widely used as a base for the performance of other active learning techniques. However, CAL is not practical in terms of applying to most classification methods due to its requirement to find the classifiers that disagree on a given sample. As a result, it is mainly used for theoretical analysis and not in practical implementations. For details of CAL, please refer to [Hanneke 2013].

For CAL, we can derive this conclusion [Hanneke 2013]:

Lemma 2 *If more than N samples are given, where $N = \theta(\epsilon)\big(\log(\theta(\epsilon)) + \log(\frac{\log(1/\epsilon)}{\delta})\big)\log(1/\epsilon)$, then with probability at least $1-\delta$, the expected error of CAL will be bounded by ϵ, where $\theta(\epsilon)$ is a parameter depending on the underlying classification algorithm used inside the CAL active learning iterations.*

The main difficulty when computing N for CAL algorithm is how to compute the parameter $\theta(\epsilon)$ for the specific classification algorithm used inside CAL iterations. Fortunately, our LCS algorithm is based on non-linear SVM and its $\theta(\epsilon)$ can be computed based on the $\theta(\epsilon)$ result for linear separators, as provided in [Hanneke 2013]. Intuitively, for linear classifiers working on a dataset with a non-uniform distribution, the corresponding $\theta(\epsilon)$ can be computed as follows:

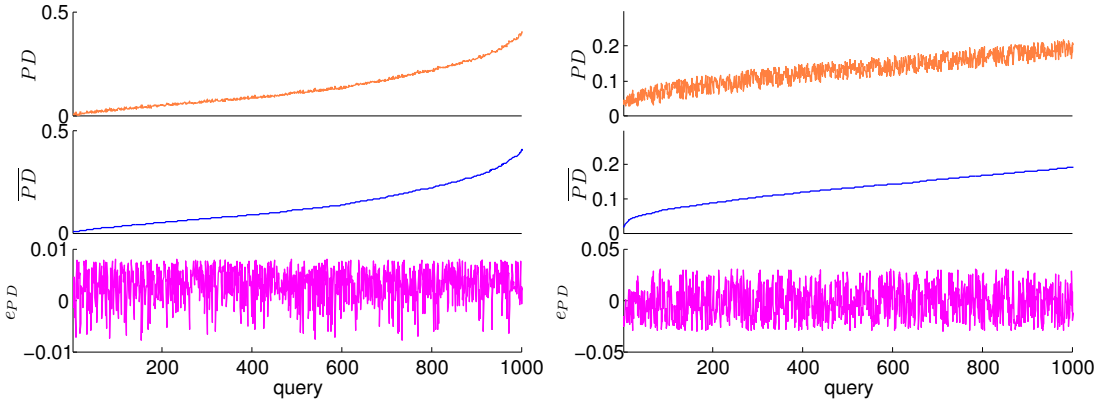
Lemma 3 *Suppose $\mathcal{C}$ is the class of k-dimensional linear separators, and the dataset distribution has a bounded density with finite support, then $\theta(\epsilon) = \mathcal{O}(1)$ for classification algorithm based on $\mathcal{C}$.*

In our LCS computation problem, the dataset is a bounded configuration space and we assume that all the points in the configuration space have the same probability to appear as samples for the active learning pipeline. Moreover, we use non-linear SVM as the underlying classification algorithm, which is essentially a linear classifier in the feature space, i.e., the mapping of the original configuration space via the kernel functions. The samples uniformly distributed in the configuration space will be mapped into samples with non-uniform distribution in the feature space. Based on Lemma 3, we can bound the $\theta(\epsilon)$ parameter corresponding to SVM by $\mathcal{O}(1)$. Next, according to Lemma 2, we can see the number of samples required by CAL active learning for our LCS computation problem is bounded by $\mathcal{O}\big(\log(\frac{\log(1/\epsilon)}{\delta})\log(1/\epsilon)\big)$, which can be further bounded by $\mathcal{O}(\log(\frac{1}{\epsilon\delta}))$ and Theorem 1 is proved. ∎

Remark: Theorem 1 tells us there is one active learning algorithm (i.e., CAL) which can produce the bound mentioned above. However, our LCS computation pipeline chooses a different exploration-and-exploitation scheme as the active learning technique, whose error bound is difficult to analyze. However, it is believed that the exploration-and-exploitation method should provide lower error bound than CAL, because 1) CAL represents the least information we could expect from a reasonable active learning algorithm (see the proof above); and 2) it is heavily used in many applications based on active learning, including computer vision [Loy et al. 2012], text classification [Tong and Koller 2002], information filtering [Zhang et al. 2003], robot grasping [Kroemer et al. 2009] and Gaussian Processes learning [Krause and Guestrin 2007]. As a result, it is reasonable to use Theorem 1 to estimate the performance of our LCS computation pipeline.

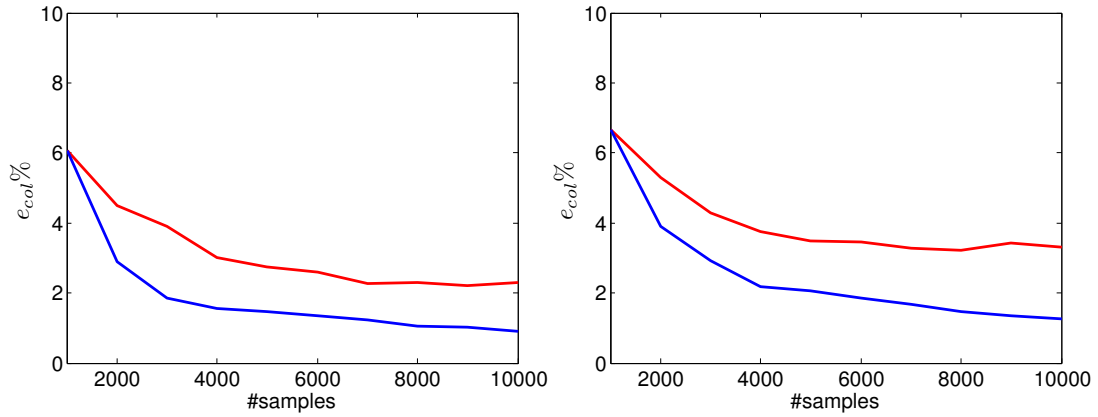
2.2 Experimental Accuracy Comparison

We have compared the relative accuracy of our algorithm with the uniform sampling methods in Figure 2 and in Figure 3.



(a) Cup-Spoon (b) Rings (Bullet)

Figure 2: *Accuracy comparison with respect to magnitude of PD. The blue, orange, and pink curves correspond to our approximate PD (i.e., $\overline{PD}$), nearly exact PD by point-based method [Lien 2009], and the error between them (e_{PD}), respectively. In these benchmarks, we observe less than 3% relative error in PD computation using our algorithm.*



(a) e_{col} for 3D Bunny Models (b) e_{col} for 3D Dragon Models

Figure 3: *Relative error convergence of active learning (blue) vs. uniform sampling (red) for complex 3D object pairs. These results demonstrate the benefits of active learning in terms of fewer samples and improved accuracy. This convergence bound is similar to the one derived from Theorem 1.*

3 Supplemental Results

We have used many benchmarks to evaluate the performance of our algorithm. Most of them are shown in Figure 1 in the paper and some additional simple models are shown in Figure 4 of this supplemental report (also refer to the accompany video). In this section, we highlight the performance of our algorithm on many 2D and 3D benchmarks.

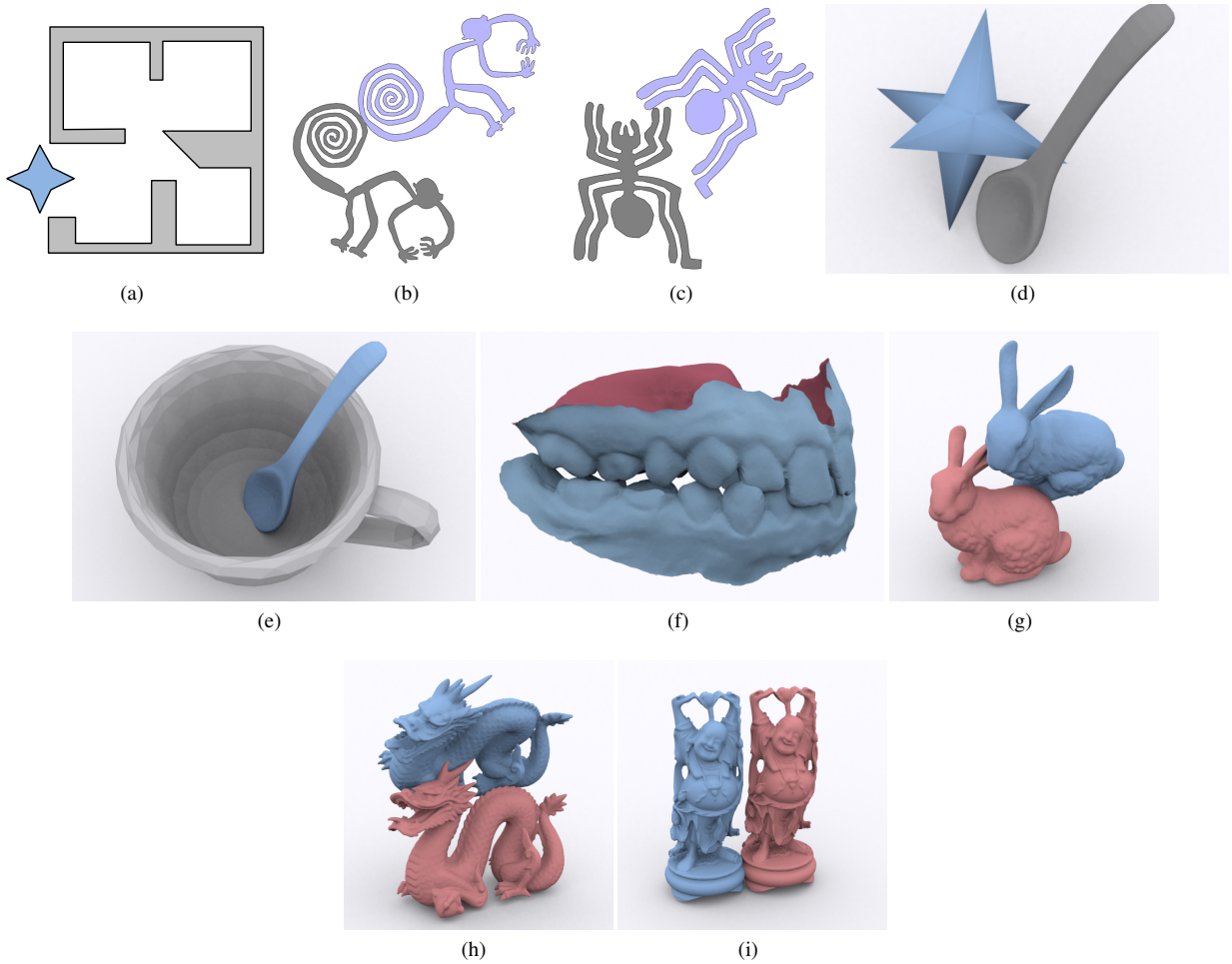


Figure 4: *Some additional simple 2D and 3D models used in robotics and dynamic simulation for evaluating the performance of our PD algorithm in Table 1 in the paper. Other complex models are shown in Figure 1 in the paper. 2D star, room, monkey and spider models consist of 8, 25, 148, 250 edges, respectively. 3D star, cup, spoon, upper teeth, lower teeth, bunny and dragon models consist of 24, 1000, 1344, 47K, 40K, 70K, 230K triangles, respectively. We use Buddha models with different complexities, varying from 20K to 1M triangles.*

3.1 AngryBirds using Box2D

Box2D uses PD computation in impulse-based collision response algorithm. We demonstrate the performance of our algorithm on a complex benchmark (Figure 6), angry bird characters falling into a complex chute. We precompute the LCS approximation in 3-DOF $\mathcal{C}$-space. For comparison, we used a local PD algorithm based on convex decomposition. The convex decomposition results in 17, 30, and 32 convex pieces for Red Bird, White Bird and Green Piggy models, respectively. 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 5 and Figure 8).

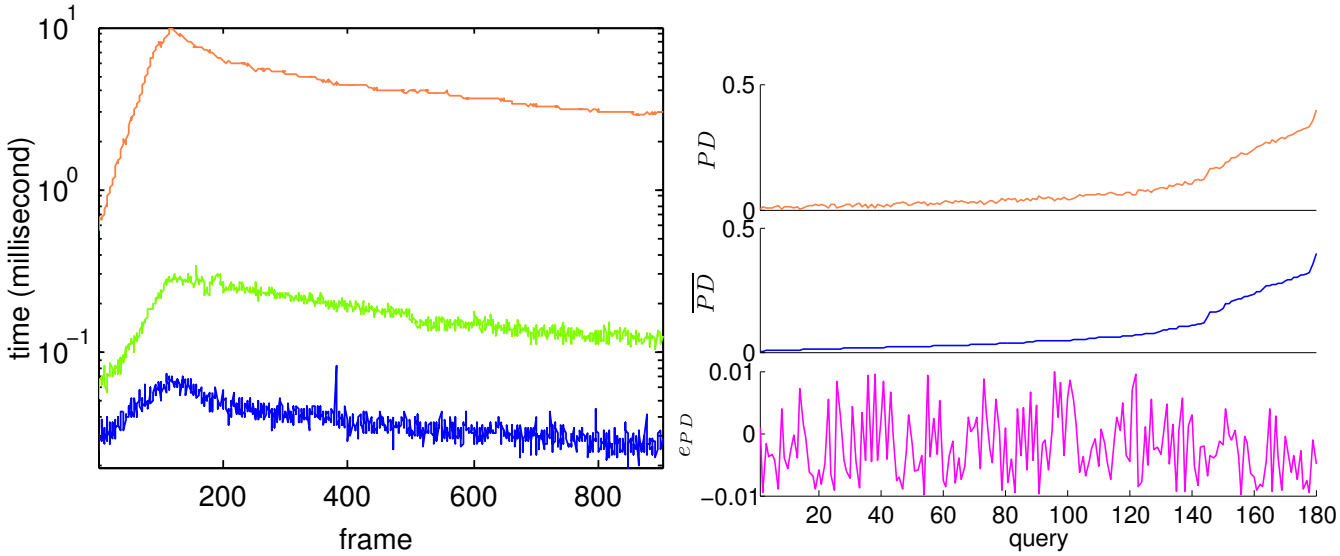
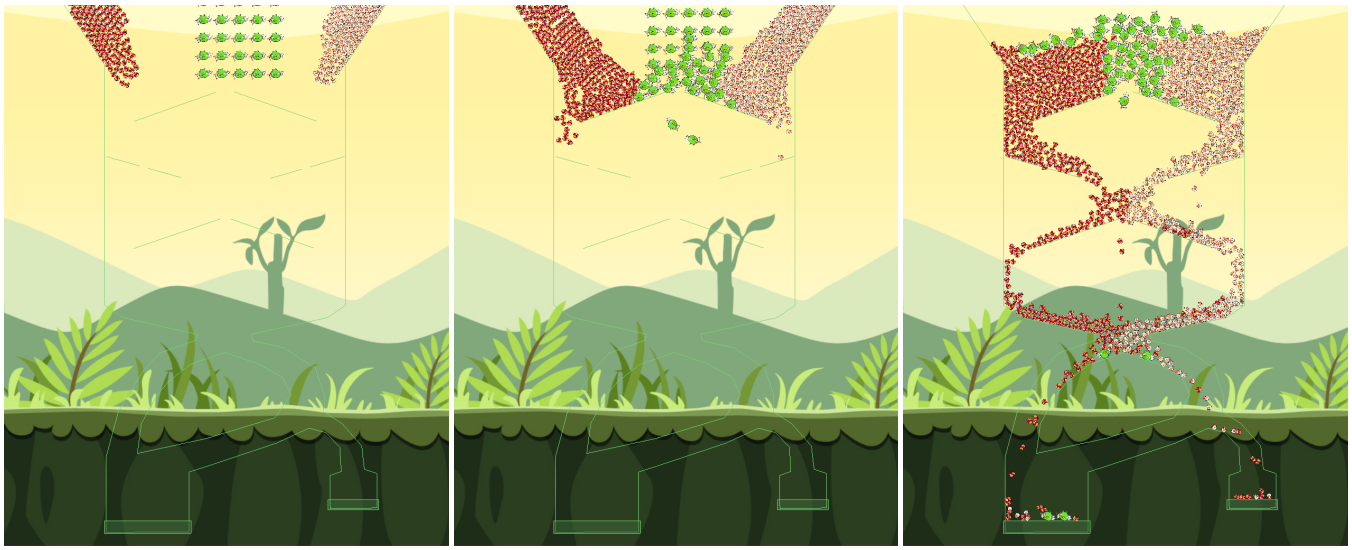
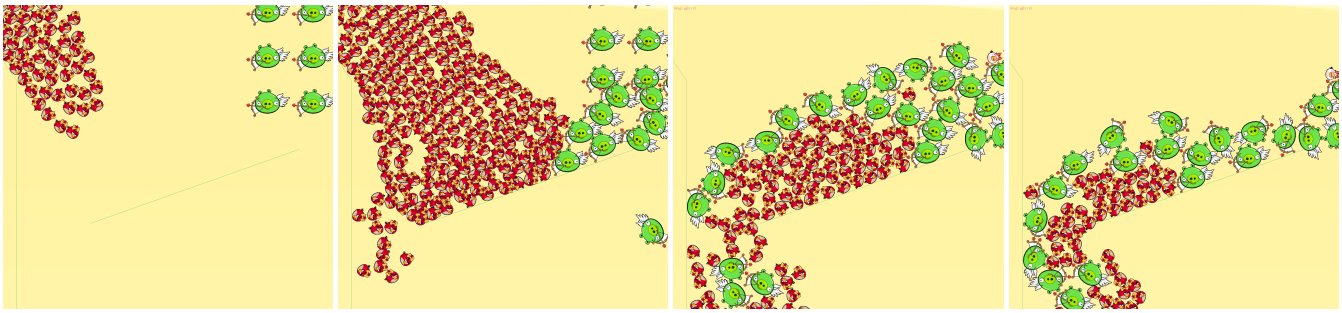


Figure 5: *(a) Relative performance of PD computation for the Angry Birds benchmark. The blue curve represents the query time computed by our algorithm. The green curve corresponds to the query time computed using convex decomposition and local PD between convex pairs. The orange curve represents the query time computed using exact (sliced) Minkowski sum (2D objects). (b) Accuracy comparison with respect to magnitude of PD. We show the values for different PD queries performed during a single frame of the simulation. The blue, orange, and pink curves correspond to our approximate PD (i.e., $\overline{PD}$), exact PD, and the error between them (e_{PD}), respectively. In these benchmarks, we observe less than 3% relative error in PD computation using our algorithm.*

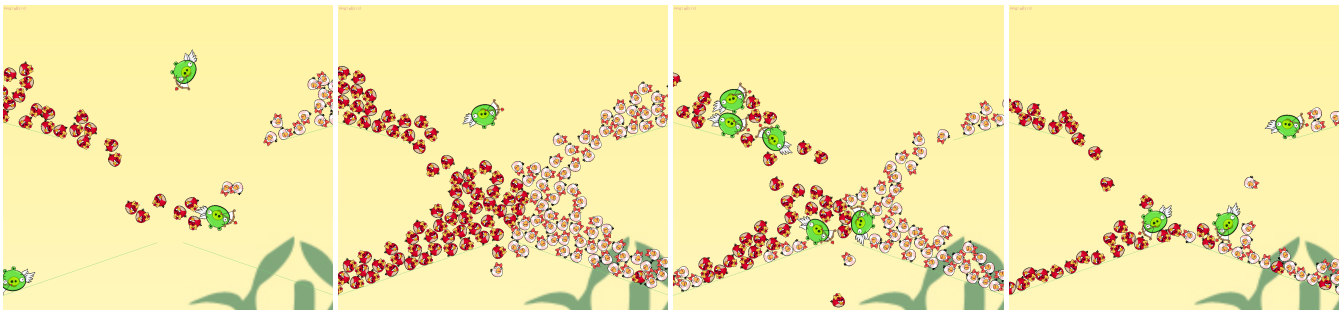
(a) Characters



(b) View 1



(c) View 2



(d) View 3

Figure 6: *Angry Birds characters and its dynamic simulation using Box2D. We have observe nearly 20 times improvement in PD query using our active learning algorithm over techniques based on convex decomposition used in Box2D. Our algorithm avoid the annoying cases where local PD algorithms cant separate the non-convex objects with deep inter-penetration.*

3.2 Nazca Spiders in A Tumbler

Box2D uses PD computation in impulse-based collision response algorithm. We demonstrate the performance of our algorithm on 2D non-convex Nazca spiders rolling in a tumbler. We precompute the LCS approximation in 3-DOF $\mathcal{C}$-space. We compared our algorithm with the convex decomposition-based algorithm in which a Nazca spider is decomposed into 77 convex pieces. We have observe nearly 20 times improvement in PD query using our active learning algorithm over techniques based on the convex decomposition algorithm used in Box2D (see Figures 7 and 8).

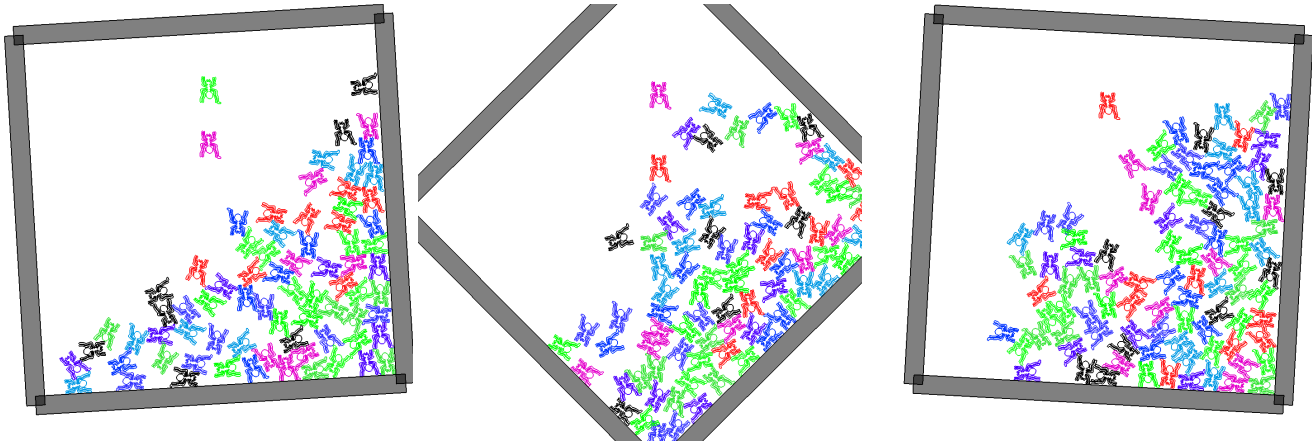
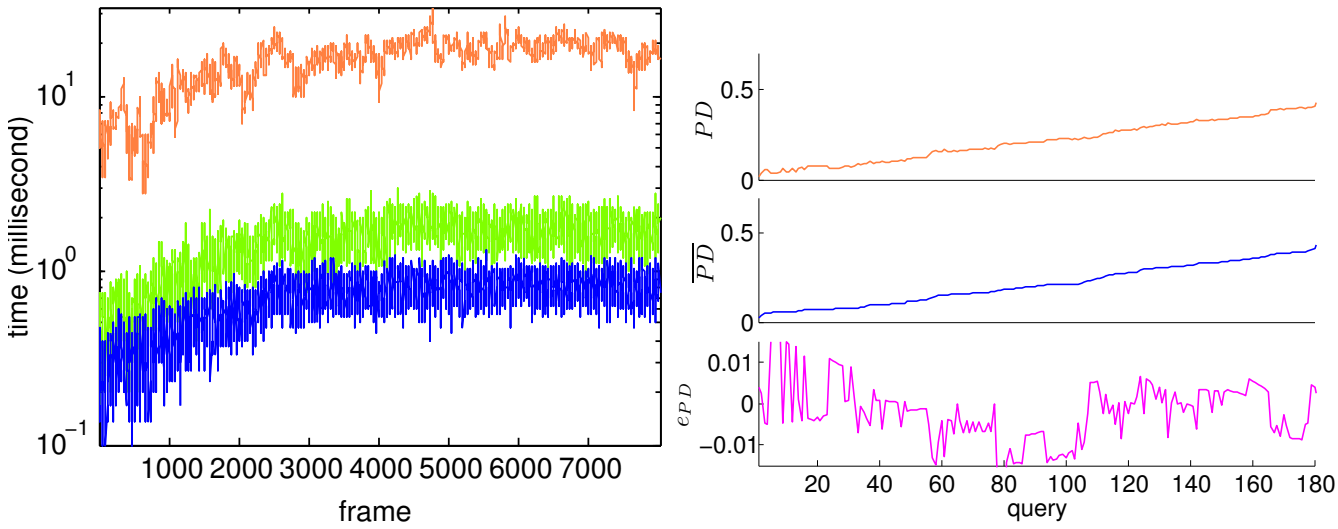


Figure 7: *Spiders in a tumbler and its dynamic simulation using Box2D. .*

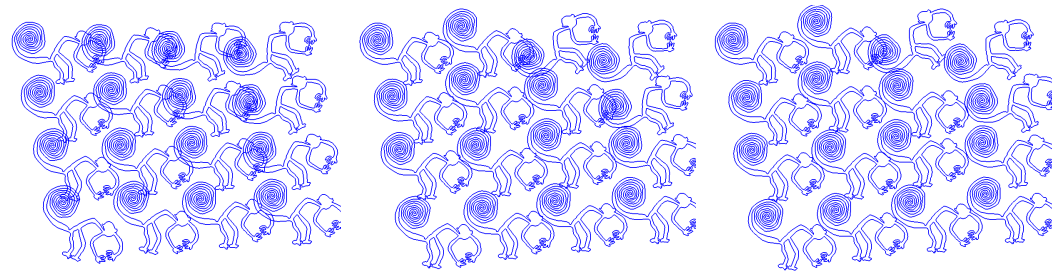


(a) Performance (b) Errors

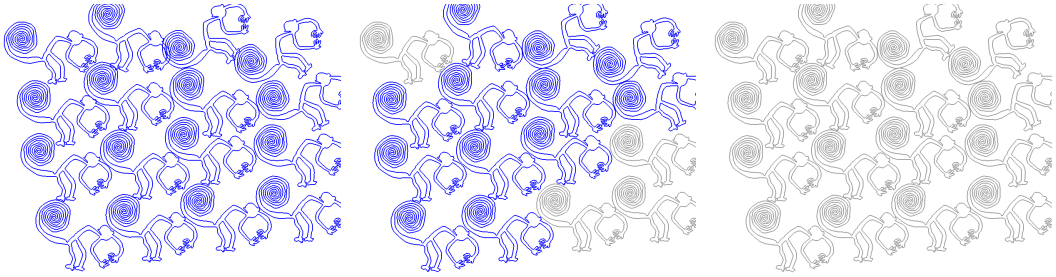
Figure 8: *Performance and Errors. (a) Relative performance of PD computation for the Rolling Nazca Spiders benchmark. The blue curve represents the query time computed by our algorithm. The green curve corresponds to the query time computed using convex decomposition and local PD between convex pairs. The orange curve represents the query time computed using exact (sliced) Minkowski sum (2D objects). (b) Accuracy comparison with respect to magnitude of PD. We show the values for different PD queries performed during a single frame of the simulation. The blue, orange, and pink curves correspond to our approximate PD (i.e., $\overline{PD}$), exact PD, and the error between them (e_{PD}), respectively. In these benchmarks, we observe less than 3% relative error in PD computation using our algorithm.*

3.3 Nazca Monkeys

We show the Nazca monkey simulation and compare the global translational PD (Figure 9) with local translational PD (Figure 10). Local translational PD corresponding to each isolated contact region does not separate the overlapping objects with deep inter-penetration.

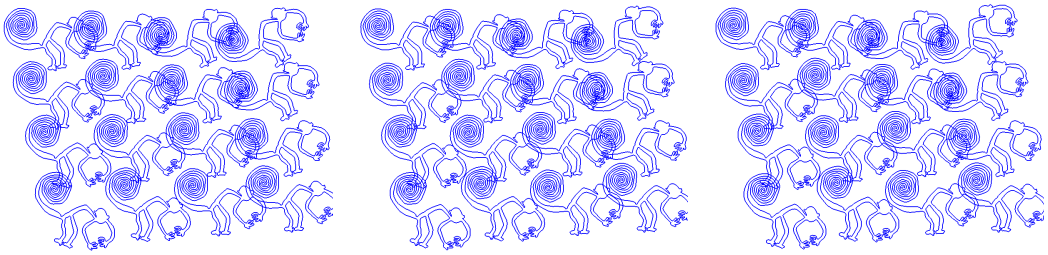


(a) frame=0 (b) frame=5 (c) frame=10

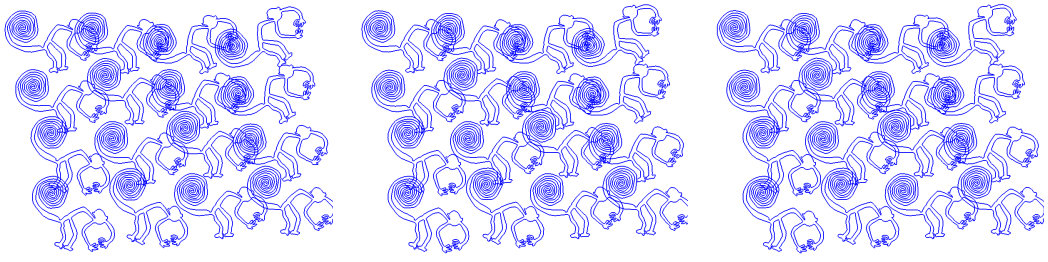


(d) frame=20 (e) frame=37 (f) frame=40

Figure 9: *Nazca monkey simulation using global PD. Here show frames 0, 5, 10, 20, 37 and 40. After a few simulation steps, all the intersecting objects are separate. Intersecting objects are displayed in blue and separate objected are displayed in grey.*



(a) frame=0 (b) frame=5 (c) frame=10

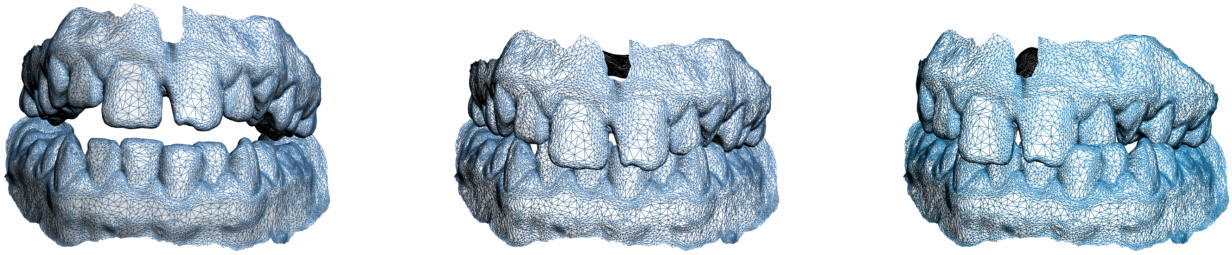


(d) frame=20 (e) frame=37 (f) frame=40

Figure 10: *Nazca monkey simulation using local PD. Here show frames 0, 5, 10, 20, 37 and 40. Local translational PD corresponding to each isolated contact region cant separate the non-convex objects with deep inter-penetration.*

3.4 Lower Teeth vs. Upper Teeth

Figure 11 shows a pair of moving teeth, where each jaw is composed of more than 40,000 triangles. We generated a sequence of motions for the upper jaw (also shown in the accompanying video). It exhibits inter-penetration and multiple contacts. We tested our PD algorithm based on LCS approximation in 6-DOF $\mathcal{C}$-space. As compared with the convex decomposition-based algorithm, we observe more than 10 times performance improvement (see Figure 12)



(a) (b) (c)

Figure 11: *A sequence of motions for the upper jaw.*

Figure 12 compares the query performance between our method and the convex-decomposition-based approach.

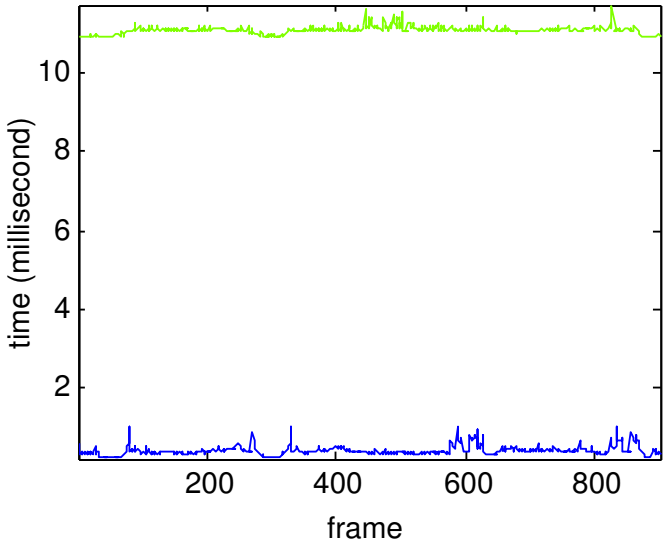
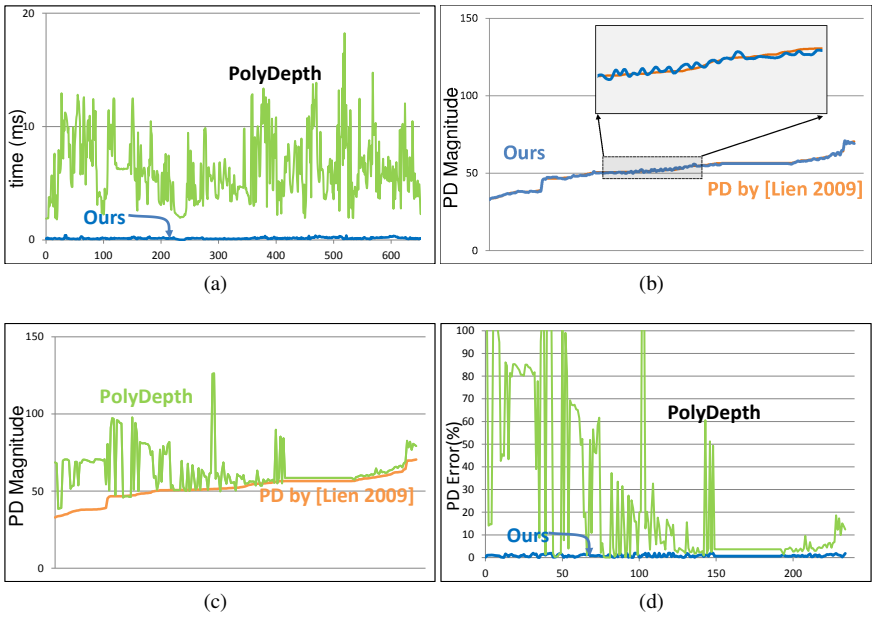


Figure 12: *Relative performance of PD computation on teeth benchmark: The blue curve represents the query time computed by our algorithm. The green curve corresponds to the query time computed using convex decomposition and local PD between convex pairs.*

3.5 Buddha vs. Buddha

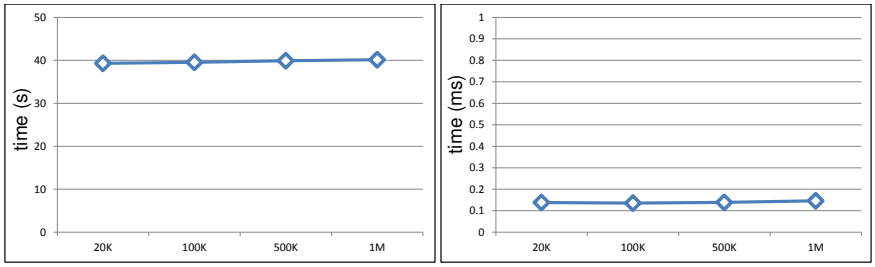
In this benchmark, we use different Buddha model complexity varying from $20K$ triangles to $1M$ triangles, generated by repeatedly subdividing the $20K$ model. Figure 13 shows the performance and accuracy comparison with PolyDepth [Je et al. 2012] and the timings of PD_t runtime query. We observe lower query time and higher accuracy in our benchmark. Figure 14 shows the performance scalability with respect to the model complexity.



(a) (b)

(c) (d)

Figure 13: *The performance and accuracy comparison with PolyDepth [Je et al. 2012] on Buddha-Buddha benchmark with 20K triangles. (a) computational time (on average, 0.10ms based on our algorithm vs. 7.15ms 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 our algorithm vs. PolyDepth; (d) our global PD algorithm (blue) has lower error as compared to PolyDepth that performs local optimization.*



(a) Precomputing LCS (b) Runtime Query

Figure 14: *The performance scalability on Buddha-Buddha benchmark with model complexity varying from $20K$ triangles to $1M$ triangles. (a) time of active learning; (b) time of runtime query.*

4 LCS approximation

In this section, we demonstrate the performance of our LCS approximation algorithm on many 2D and 3D benchmarks. In all our benchmarks, we are able to compute a good approximation in a few iterations.

4.1 Star vs. Room

In Figure 15, we show the LCS sequences generated when using active learning to gradually construct the approximate contact space between 2D non-convex shapes (star and room) given in Figure 3 in the paper. We show the approximation after i-th iteration and the number of support vectors. We can see that the final result successfully captures the topological structure of the exact $\mathcal{C}_{cont}$.

(a) $LCS_0, |S| = 37$ (b) $LCS_1, |S| = 42$ (c) $LCS_2, |S| = 49$ (d) $LCS_3, |S| = 60$

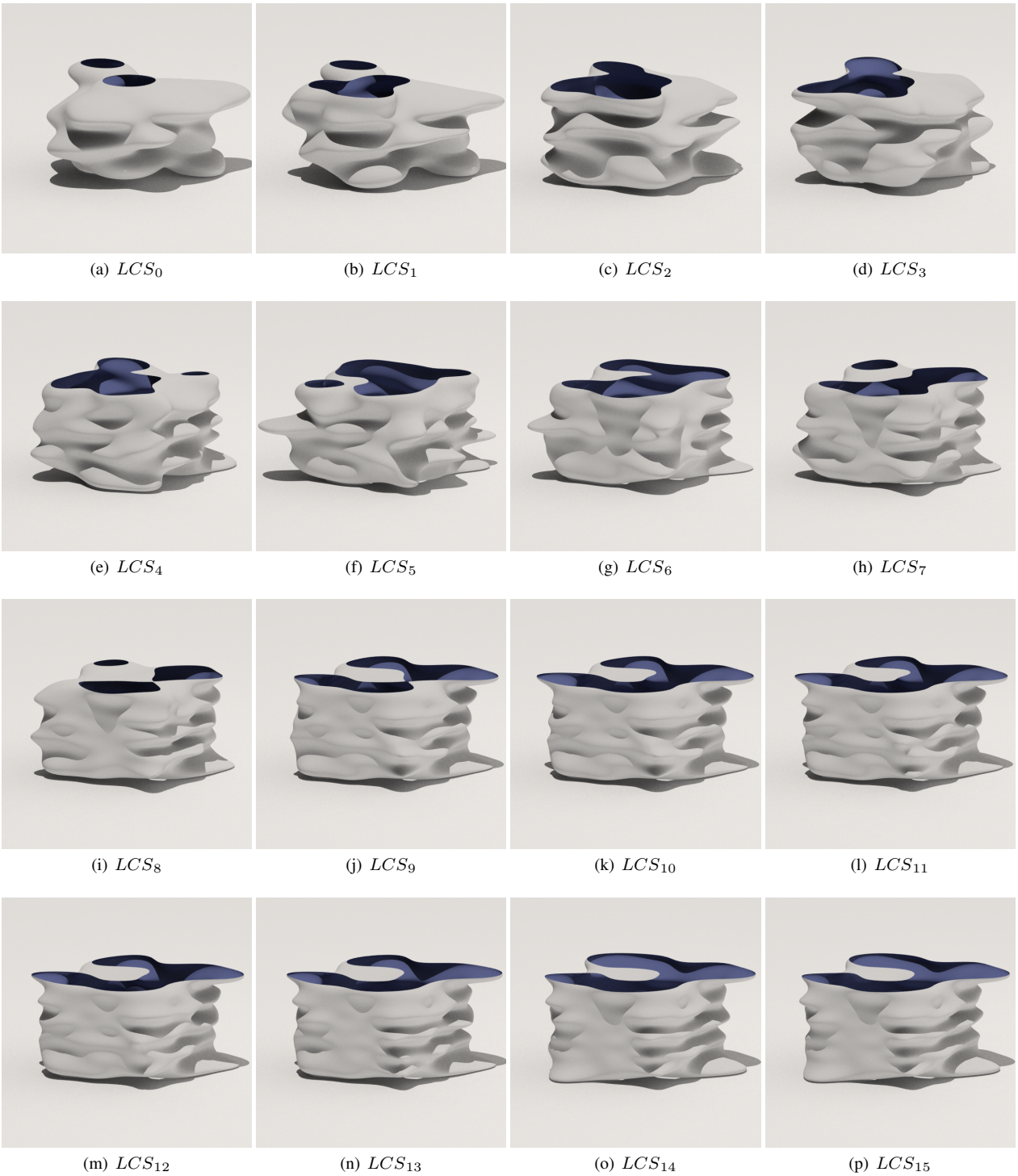
(e) $LCS_4, |S| = 75$ (f) $LCS_5, |S| = 96$ (g) $LCS_6, |S| = 124$ (h) $LCS_7, |S| = 150$

(i) $LCS_8, |S| = 198$ (j) $LCS_9, |S| = 216$ (k) $LCS_{10}, |S| = 255$ (l) $LCS_{12}, |S| = 298$

(m) $LCS_{14}, |S| = 327$ (n) $LCS_{16}, |S| = 403$ (o) $LCS_{18}, |S| = 454$ (p) $LCS_{20}, |S| = 654$

Figure 15: *LCS computation using active learning for PD_t query between 2D star and room (both are non-convex shapes) given in Figure 2 in the paper. We are able to compute a good approximation to the LCS in about 10 iterations.*

In Figure 16, we show the LCS sequences for PD_g computation.



(a) LCS_0 (b) LCS_1 (c) LCS_2 (d) LCS_3

(e) LCS_4 (f) LCS_5 (g) LCS_6 (h) LCS_7

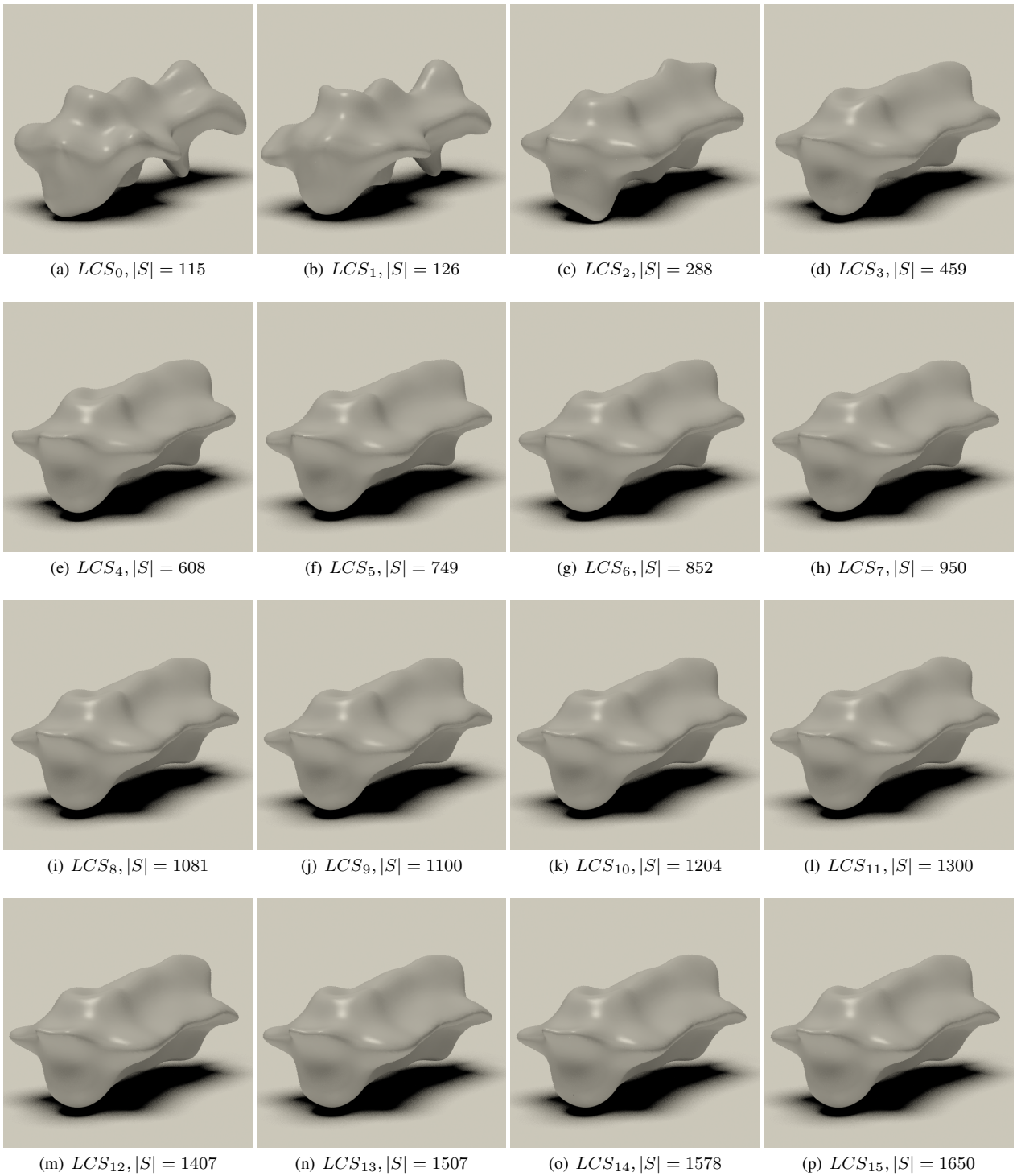
(i) LCS_8 (j) LCS_9 (k) LCS_{10} (l) LCS_{11}

(m) LCS_{12} (n) LCS_{13} (o) LCS_{14} (p) LCS_{15}

Figure 16: *LCS computation using active learning for PD_g query between star and room models given in Figure 2 in the paper. We are able to compute a good approximation to the LCS in about 10 iterations*

4.2 Star vs. Spoon

Figure 17 shows the LCS sequences generated when using active learning to gradually construct the approximate contact space between 3D star and spoon models. We can see that the LCS converges rather quickly. This demonstrates the benefits of active learning.



(a) $LCS_0, |S| = 115$ (b) $LCS_1, |S| = 126$ (c) $LCS_2, |S| = 288$ (d) $LCS_3, |S| = 459$

(e) $LCS_4, |S| = 608$ (f) $LCS_5, |S| = 749$ (g) $LCS_6, |S| = 852$ (h) $LCS_7, |S| = 950$

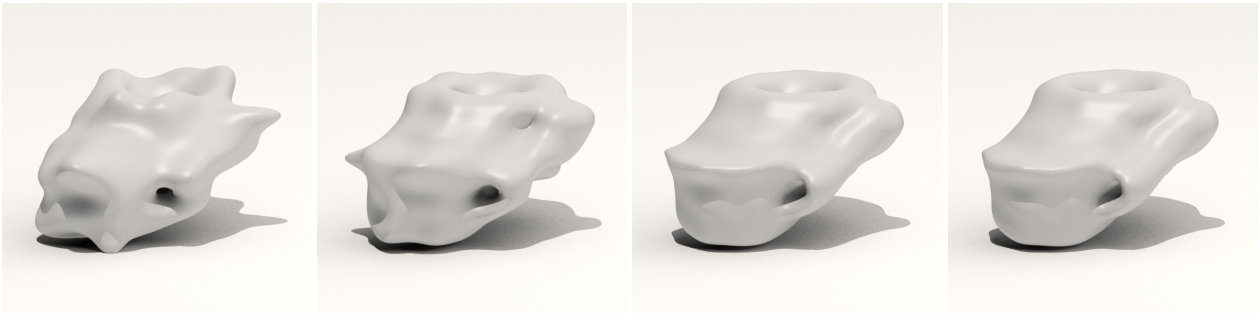
(i) $LCS_8, |S| = 1081$ (j) $LCS_9, |S| = 1100$ (k) $LCS_{10}, |S| = 1204$ (l) $LCS_{11}, |S| = 1300$

(m) $LCS_{12}, |S| = 1407$ (n) $LCS_{13}, |S| = 1507$ (o) $LCS_{14}, |S| = 1578$ (p) $LCS_{15}, |S| = 1650$

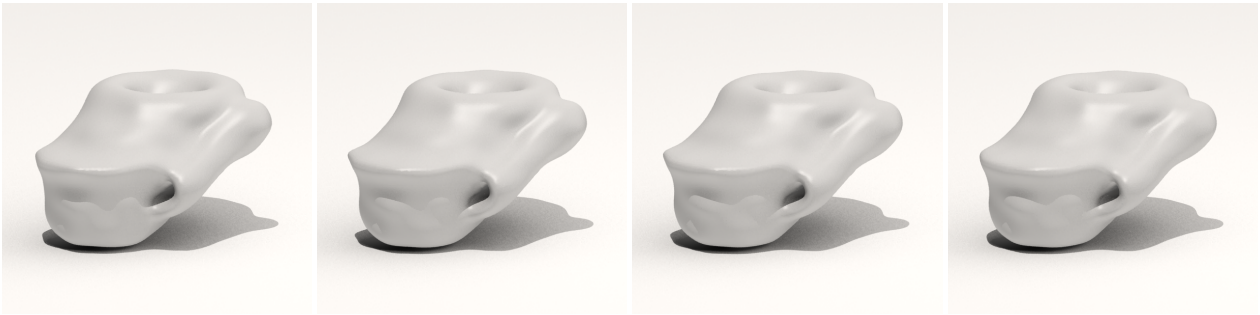
Figure 17: *LCS computation using active learning for PD_t query between 3D star and spoon models.*

4.3 Cup vs. Spoon

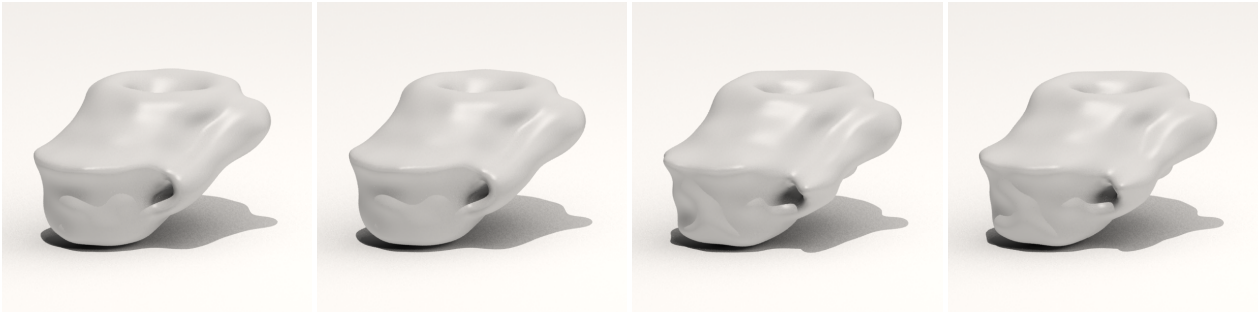
Figure 18 shows the LCS sequences generated when using active learning to gradually construct the approximate contact space between 3D cup and spoon models. We can see that the LCS converges rather quickly. This demonstrates the benefits of active learning.



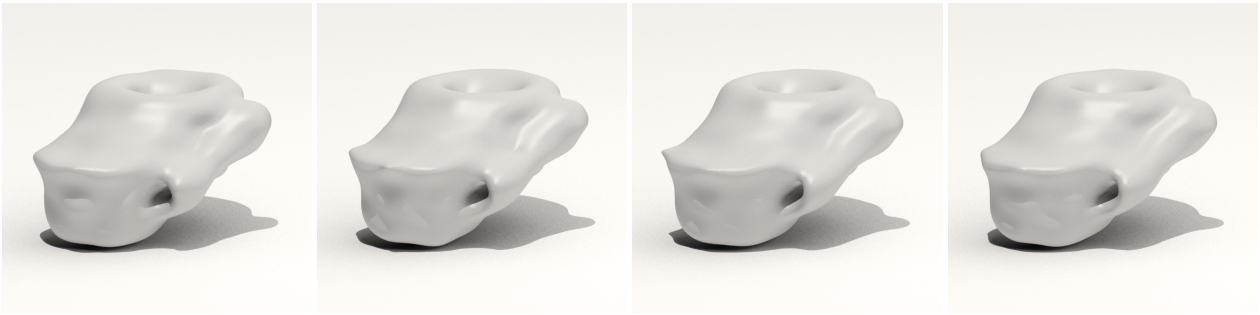
(a) $LCS_0, |S| = 231$ (b) $LCS_1, |S| = 234$ (c) $LCS_2, |S| = 385$ (d) $LCS_3, |S| = 563$



(e) $LCS_4, |S| = 733$ (f) $LCS_5, |S| = 869$ (g) $LCS_6, |S| = 999$ (h) $LCS_7, |S| = 1121$



(i) $LCS_8, |S| = 1230$ (j) $LCS_9, |S| = 1350$ (k) $LCS_{10}, |S| = 1361$ (l) $LCS_{11}, |S| = 1491$

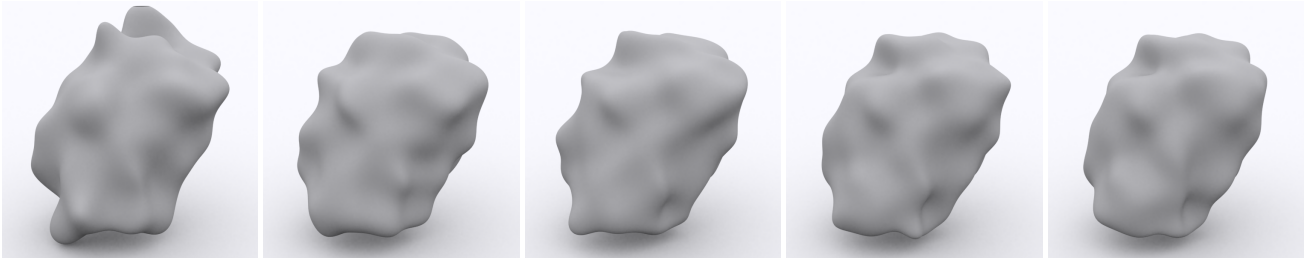


(m) $LCS_{12}, |S| = 1572$ (n) $LCS_{13}, |S| = 1670$ (o) $LCS_{14}, |S| = 1761$ (p) $LCS_{15}, |S| = 1848$

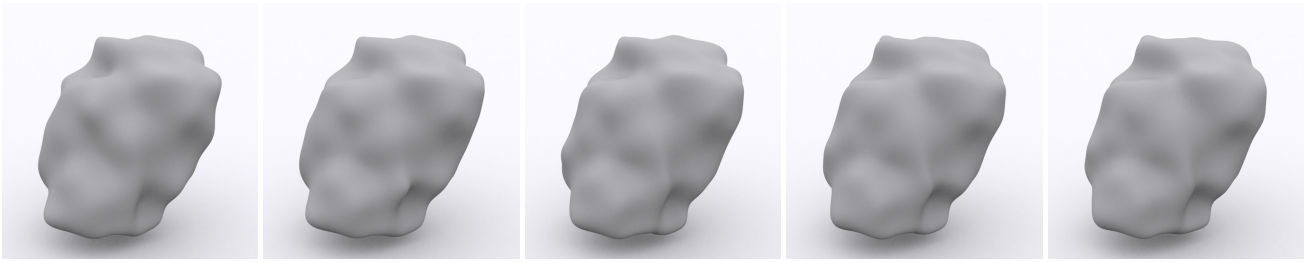
Figure 18: *LCS computation using active learning for PD_t query between 3D cup and spoon models.*

4.4 Bunny vs. Bunny

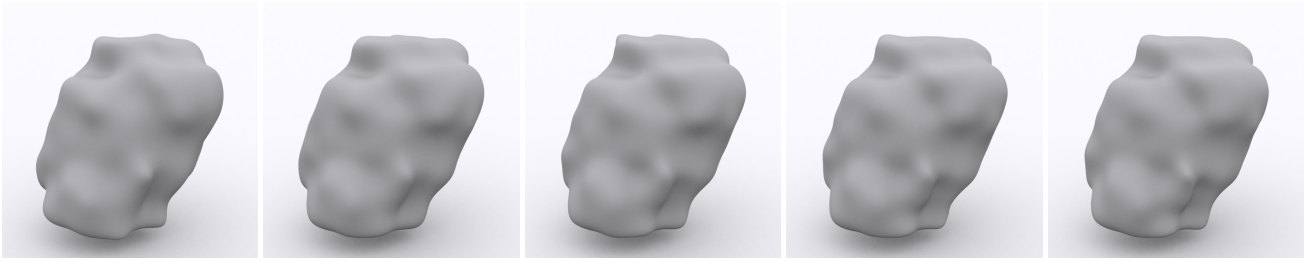
Figure 19 shows the LCS sequences generated when using active learning to gradually construct the approximate contact space between 3D bunny models. We can see that the LCS converges very fast due to active learning.



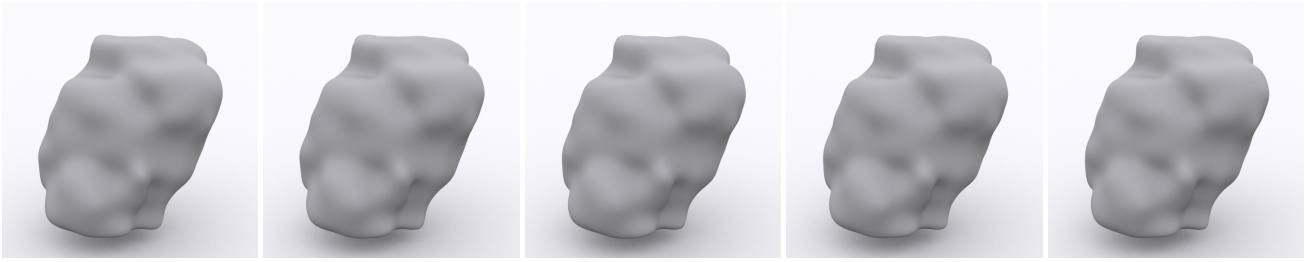
(a) $LCS_0, |S| = 237$ (b) $LCS_1, |S| = 267$ (c) $LCS_2, |S| = 319$ (d) $LCS_3, |S| = 367$ (e) $LCS_4, |S| = 421$



(f) $LCS_5, |S| = 471$ (g) $LCS_6, |S| = 521$ (h) $LCS_7, |S| = 562$ (i) $LCS_8, |S| = 617$ (j) $LCS_9, |S| = 666$



(k) $LCS_{10}, |S| = 725$ (l) $LCS_{11}, |S| = 793$ (m) $LCS_{12}, |S| = 871$ (n) $LCS_{13}, |S| = 927$ (o) $LCS_{14}, |S| = 1016$

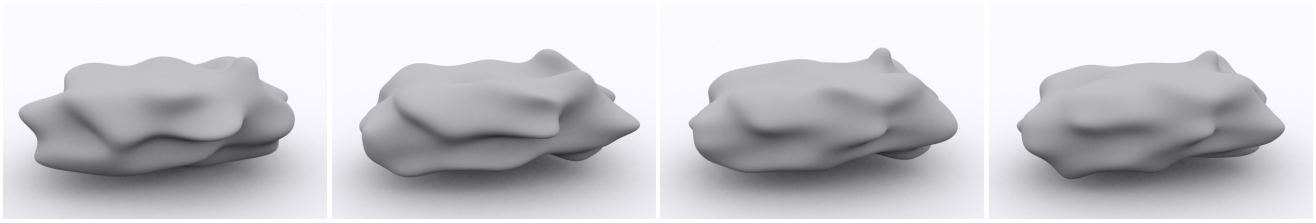


(p) $LCS_{15}, |S| = 1123$ (q) $LCS_{16}, |S| = 1293$ (r) $LCS_{17}, |S| = 1338$ (s) $LCS_{18}, |S| = 1450$ (t) $LCS_{19}, |S| = 1517$

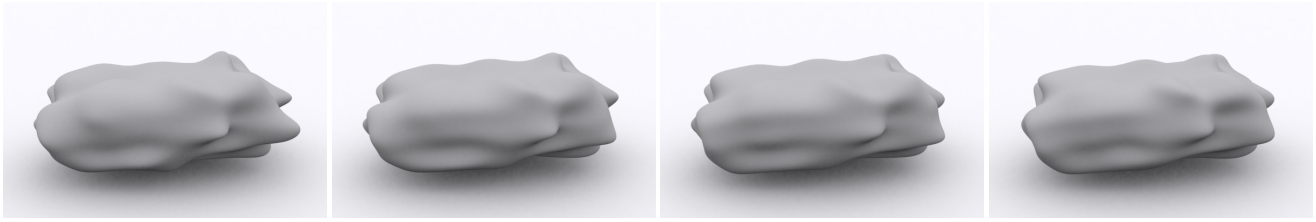
Figure 19: *LCS computation using active learning for PD_t query between 3D bunny models. Each bunny model consists of $70K$ triangles.*

4.5 Dragon vs. Dragon

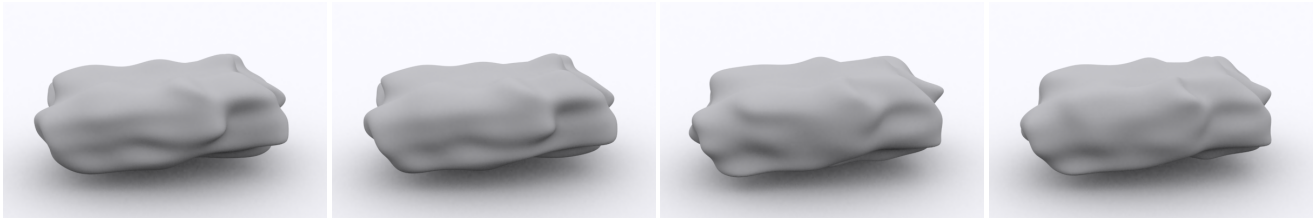
Figure 20 shows the LCS sequences generated when using active learning to gradually construct the approximate contact space between 3D dragon models. We can see that the LCS converges very fast due to active learning.



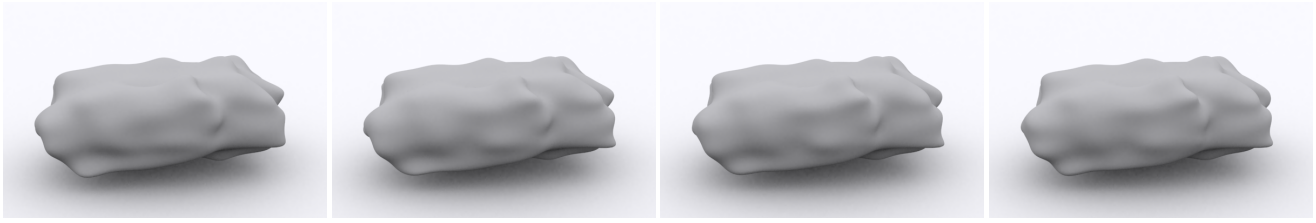
(a) $LCS_0, |S| = 243$ (b) $LCS_1, |S| = 324$ (c) $LCS_2, |S| = 374$ (d) $LCS_3, |S| = 454$



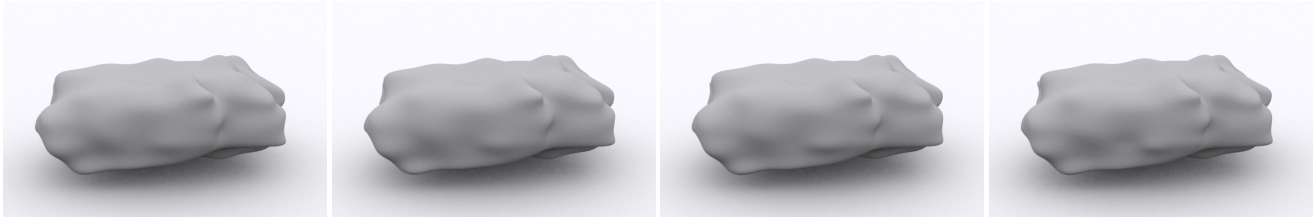
(e) $LCS_4, |S| = 528$ (f) $LCS_5, |S| = 570$ (g) $LCS_6, |S| = 724$ (h) $LCS_7, |S| = 752$



(i) $LCS_8, |S| = 790$ (j) $LCS_9, |S| = 854$ (k) $LCS_{10}, |S| = 876$ (l) $LCS_{11}, |S| = 943$



(m) $LCS_{12}, |S| = 1023$ (n) $LCS_{13}, |S| = 1045$ (o) $LCS_{14}, |S| = 1172$ (p) $LCS_{15}, |S| = 1232$

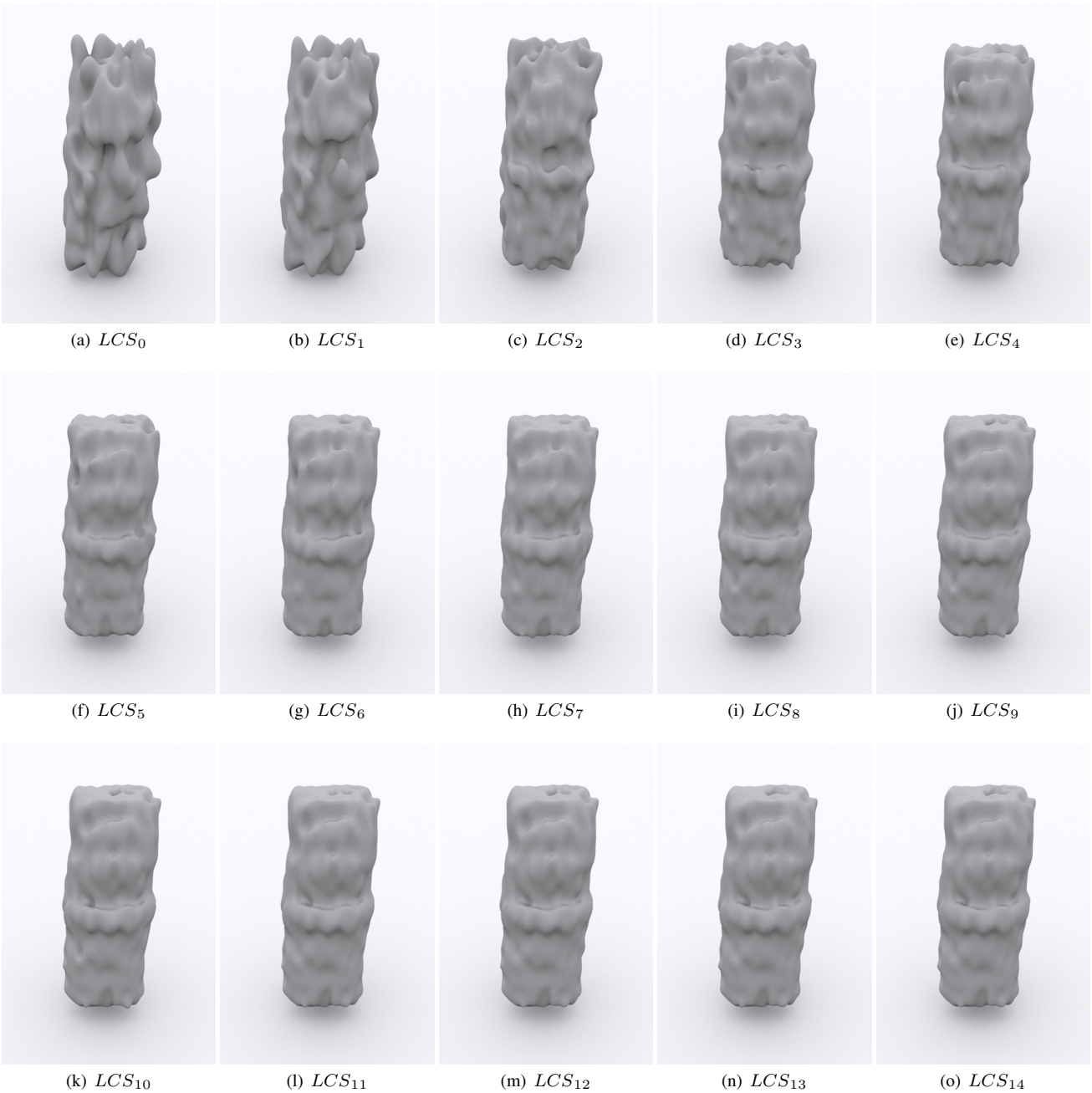


(q) $LCS_{16}, |S| = 1304$ (r) $LCS_{17}, |S| = 1413$ (s) $LCS_{18}, |S| = 1490$ (t) $LCS_{19}, |S| = 1532$

Figure 20: *LCS computation using active learning for PD_t query between 3D dragon models. Each dragon model consists of $230K$ triangles.*

4.6 Buddha vs. Buddha

Figure 21 shows the LCS sequences generated when using active learning to gradually construct the approximate contact space between 3D Buddha models. We can see that the LCS converge very fast.



(a) LCS_0 (b) LCS_1 (c) LCS_2 (d) LCS_3 (e) LCS_4

(f) LCS_5 (g) LCS_6 (h) LCS_7 (i) LCS_8 (j) LCS_9

(k) LCS_{10} (l) LCS_{11} (m) LCS_{12} (n) LCS_{13} (o) LCS_{14}

Figure 21: *LCS computation using active learning for PD_t query between 3D Buddha models. The complexity of the Buddha model varies from $20K$ to $1M$ triangles.*

References

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

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

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.

KRAUSE, A., AND GUESTRIN, C. 2007. Nonmyopic active learning of gaussian processes: an exploration-exploitation approach. In *Proceedings of International Conference on Machine learning*, 449–456.

KROEMER, O., DETRY, R., PIATER, J., AND PETERS, J. 2009. Active learning using mean shift optimization for robot grasping. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2610–2615.

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.

LOY, C., HOSPEDALES, T., XIANG, T., AND GONG, S. 2012. Stream-based joint exploration-exploitation active learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1560–1567.

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

VAPNIK, V. N. 1995. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA.

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.

ZHANG, Y., XU, W., AND CALLAN, J. 2003. Exploration and exploitation in adaptive filtering based on bayesian active learning. In *Proceedings of International Conference on Machine Learning*, 896–903.

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