Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

Local object crop collision network for efficient simulation of non-convex objects in GPU-based simulators

Dongwon Son and Beomjoon Kim

***Abstract*—Our goal is to develop an efficient contact detection algorithm for large-scale GPU-based simulation of non-convex objects. Current GPU-based simulators such as IsaacGym [16] and Brax [11] must trade-off speed with fidelity, generality, or both when simulating non-convex objects. Their main issue lies in contact detection (CD): existing CD algorithms, such as Gilbert–Johnson–Keerthi (GJK), must trade off their computational speed with accuracy which becomes expensive as the number of collisions among non-convex objects increases. We propose a data-driven approach for CD, whose accuracy depends only on the quality and quantity of offline dataset rather than online computation time. Unlike GJK, our method inherently has a uniform computational flow, which facilitates efficient GPU usage based on advanced compilers such as XLA (Accelerated Linear Algebra) [2]. Further, we offer a data-efficient solution by learning the patterns of colliding local crop object shapes, rather than global object shapes which are harder to learn. We demonstrate our approach improves the efficiency of existing CD methods by a factor of 5-10 for non-convex objects with comparable accuracy. Using the previous work on contact resolution for a neural-network-based contact detector [24], we integrate our CD algorithm into the open-source GPU-based simulator, Brax, and show that we can improve the efficiency over IsaacGym and generality over standard Brax. We highly recommend the videos of our simulator included in the supplementary materials. https://sites.google.com/view/locc-rss2023/home**

I. Introduction

With an ever-increasing demand for bigger datasets, GPU-based simulators are becoming an essential tool in robotics. Unlike CPU-based simulators, they can simulate thousands of environments in parallel, which makes big data generation extremely efficient. In fact, several works in robot manipulation and locomotion have empirically demonstrated that by utilizing GPU-based simulators, you attain a significant training speed advantage over CPU-based simulators [16, 11].

While parallelism improves efficiency over CPU-based simulators, the current state-of-the-art GPU-based simulators, like IsaacGym [16], slow down significantly when simulating non-convex objects as the number of environments grows. This is well demonstrated in Figure 2 (left). The simulation time of IG-SPHERE, which simulates dropping spheres in IsaacGym, remains more or less similar even as we increase the number of environments. However, when you simulate *non*-convex objects, the speed starts to deteriorate rapidly with respect to the number of environments, as demonstrated by IG-CVXD, which approximates non-convex objects using convex decomposition [17].

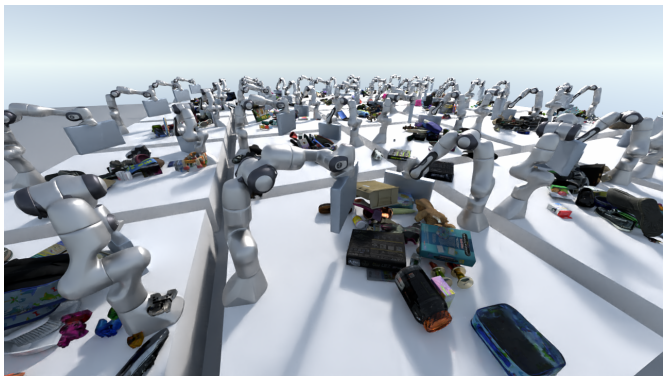


Fig. 1: Snapshot of parallel simulation with BRAX-LOCC, where two Franka Emika PANDA robots are pushing 25 objects most of which are non-convex. Qualitatively, it robustly simulates heavy interactions between non-convex objects. We recommend the readers check the video included in the supplementary material.

The fundamental reason for the slowdown in IsaacGym is CD. The most widely-used CD algorithm, GJK [12], only works with convex objects and needs convex decomposition for non-convex objects. So to improve its accuracy, we need to increase the number of elements in the decomposition, but this significantly increases online computational time due to an increase in the pairs of elements for which we need to check the collisions.

Alternatively, we can use a convex hull approximation of the non-convex objects, denoted IG-CVXH, whose speed also remains more or less constant with respect to the number of environments as shown in Figure 2 (left). However, the degradation in simulation fidelity is typically too high to bear, as shown in the approximated collision mesh of objects in Figure 2 (right).

Furthermore, GJK involves branching, such as if-statements, and does not have a uniform computational flow (UCF). As a result, it is difficult to benefit from the advanced optimization

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST)) and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00311, Development of Goal-Oriented Reinforcement Learning Techniques for Contact-Rich Robotic Manipulation of Everyday Objects)

The authors are with Graduate School of AI, KAIST, Seoul, Republic of Korea {dongwon.son beomjoon.kim}@kaist.ac.kr

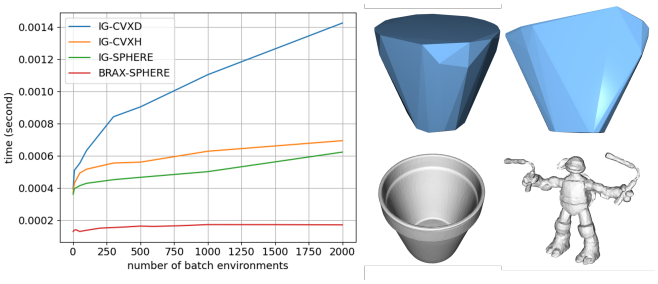


Fig. 2: (Left) Time for simulating a single time step vs. the number of environments using IsaacGym and Brax in object dropping tasks. (Right) An example of the convex hull approximation of objects used by IG-CVXH.

techniques available in domain-specific GPU compilers such as XLA [2]. This is well illustrated in the comparison of IsaacGym to Brax [1]. Brax is a GPU-based simulator that, unlike IsaacGym, has a UCF for all of its computations. When we compare Brax and IsaacGym in simulating dropping spheres, Figure 2 (left) shows that the speed of BRAX-SPHERE is much higher than IG-SPHERE. However, this comes at a cost in generality: Brax is limited to simulating only convex objects, due to the lack of contact detection algorithms that have a UCF *and* handle general shapes. Furthermore, Brax requires all environments to have the same set of objects to maintain its UCF.

In [24], the authors created a contact resolution algorithm for a neural network (NN)-based contact detector. They demonstrated that their method is highly accurate and reliable, even when simulating a complex task of tightening a nut onto a bolt. However, their contact detector has a limitation in that it cannot generalize to different shapes. Our goal is to create a NN-based contact detector that can handle a variety of shapes and can easily be used within a physics engine with [24] as a contact resolution algorithm. Our approach has a performance advantage because NNs do not have branching in their computations, allowing for the use of XLA. Additionally, compared to GJK which needs to increase its online computation time for better accuracy, the accuracy of our NN-based contact detector only depends on the quality and quantity of offline collected data, while the online computation time remains constant as a NN prediction.

One simple idea to implement an NN-based collision detector is to adapt SceneCollisionNet [6] that predicts a collision between an object and the scene. That is, we first sample a point cloud from an object mesh, encode each point cloud into a shape embedding using a shape encoder, and then use the shape embeddings to predict a collision. However, this turns out to be data inefficient: because it requires the network to learn to capture the entire object shape, if we wish to generalize to a variety of objects, we would need a large number of realistic object mesh models which are expensive to obtain.

Inspired by [14, 5], we instead propose to encode only the shape of local crops of two objects that are in collision. Our intuition is that the patterns in local crops of object shapes are easier to learn than the patterns in global object shapes, as they are more frequent in data, and can be generated more cheaply. This intuition is demonstrated in Figure 3.

Our algorithm, called LOCC (Local Object Crop Collision Network), implements this idea by creating local features of object shapes. More concretely, given the poses and shapes of two objects of interest, we first define the voxel grid on the Axis-Aligned Bounding Box (AABB) each object. Then, we use a shape encoder to compute a feature for each cell of the voxel grid, and use the poses to create Oriented Bounding Box (OBB) of the voxel grid of features. We check which cells are in collision, and then pass only the features from the colliding cells to a collision predictor. Figure 4 demonstrates the computations in our LOCC.

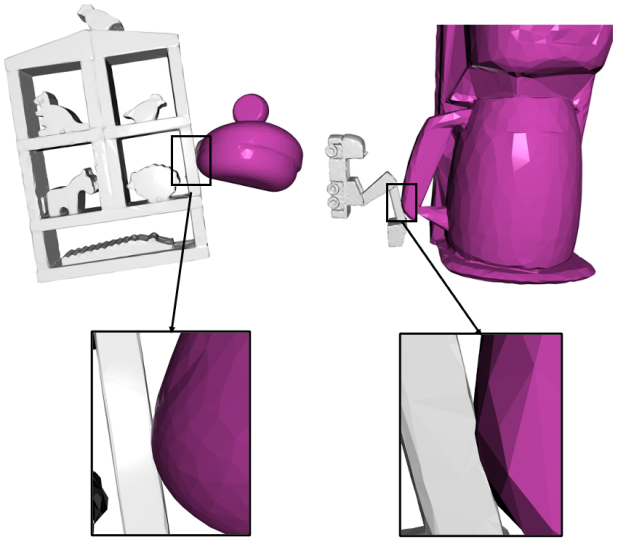


Fig. 3: Consider the figures on the top row, where two different pairs of objects are in collision. To accurately predict a collision, a naive object collision detector would need to first learn to represent the shapes of objects, but this requires many object shape data because there is a large variability in object shapes, especially for non-convex objects. Now consider the figures on the bottom row. In the local crops where collisions occur, we see that their shapes are extremely similar even though the global object shapes are totally different. Such similarity in local crops of object shapes is what we try to exploit in our algorithm to improve data efficiency.

In our experiments, we first demonstrate that LOCC outperforms state-of-the-art NN-based collision detectors, such as GLOBAL-OCN, an adaptation of [6], in terms of speed and data efficiency. Second, we implement a variant of GJK that has a UCF and show that LOCC has a 5-10 times speed gain while having a comparable accuracy. Finally, we extend Brax, the open-source GPU-based simulator, with LOCC and the contact resolution algorithm from [24] to support non-convex objects. We show that the resulting simulator, BRAX-LOCC, can simulate non-convex objects at a higher speed than

IsaacGym and more general objects than standard Brax. An example scene from BRAX-LOCC is shown in Figure 1.

II. Related Works

Collision detection using a neural network Recently, several methods have been proposed for *neural implicit functions*, which use a neural network to represent a shape. Given spatial information such as locations in 3D space, neural implicit functions output values that represent shapes [22, 14, 21, 25]. Based on these advances, several methods have been proposed for collision detection using neural implicit functions where a a set of query points is used to test if a query point lies inside of both shapes [26, 10]. However, these methods typically require a large number of query points because their accuracy depends heavily on the density of the query points. Further, explicitly reconstructing the shapes of objects, which requires a large amount of expensive shape data, has shown to be unnecessary for contact detection [6].

So instead, several methods [8, 6] propose to predict collision directly to improve data efficiency and speed. More concretely, in [6], the authors proposed a method that, given a point cloud of an object of interest, its pose, and a point cloud of a scene, determines if there is a collision between the object and the scene. In [8], the authors propose a method for learning a function for representing the configuration space obstacles which is used to check collision for a given robot configuration. Both of these approaches require learning the representation of the shape of the entire scene or object. In contrast, we learn a representation of a colliding local crops of objects to improve data efficiency.

Further, the primary purpose of [8, 6] is to improve the motion planning speed. For this, it is sufficient to determine if a collision has occurred in the current world state. However, for simulation, we not only need to detect the collisions but also determine between which objects the contact has occurred to perform contact resolution. So, we need an *object* collision detector that checks the collision between a pair of objects, rather than between a scene and an object or robot. We will refer to a NN that takes shapes and poses of two objects of interest as inputs and predicts a collision as *Object Collision Net* (OCN).

Collision resolution using an OCN The goal of contact resolution is to prevent penetration and resolve contacts so that the colliding objects move in a direction that is in accordance with the physics law after contact. In [24], the authors have proposed the method for contact resolution using an OCN. Since we use this method without modification when we integrate LOCC into Brax, we describe briefly describe how it works.

For contact resolution, we need two quantities: the set of contact points and contact forces. To find the contact points using an OCN, we first use the OCN to detect at what object pose the collision occurs. Then, we determine the directions of contact force and torque at object's center of mass by computing the gradient of the OCN with respect to the pose. The intuition is that the steepest pose direction that gets the objects out of collision is given by the gradient of the OCN, which must be in the same direction as the contact force and torque. Using these force and torque directions, we then find the contact points using the Point Isolation method [13].

Once the contact points have been determined, we define the contact constraints that enforces the colliding objects to move in the direction of not penetrating further at the contact point. The contact forces are computed using trajectory optimization subject to contact constraints and spring motion constraints based on penetration depth [19, 3]. Based on these quantities, the objects follow a simple rigid body dynamics. For more details, such as how to deal with contact resolution at the equilibrium, we refer the readers to the original paper [24].

While [24] has demonstrated the effectiveness of their contact resolution method in the challenging task of screwing a nut into a bolt, their OCN was limited to a single bolt and nut and cannot take object shapes as inputs. Our algorithm, LOCC, can be seen as a generalization of [24] to a variety of shapes.

Analytical collision detection methods and their usage in GPUs GJK [12] is used in a variety of representative physics engines, such as Havok, PhysX, and Bullet. In practice, GJK can detect collisions typically in constant time for convex objects. However, for non-convex objects, we must first make a convex decomposition of all the objects using a method such as V-HACD [18], run GJK to check collision for each pair of decomposition elements in the worst case[1], then report collision if any one of the element pairs is in a collision.

Since GPUs expedite computations by applying the same function across different environments, it is hard to efficiently use GJK on a GPU due to branching in computations. We can, as we show in our experiments, modify GJK to have a UCF. However, even in this case, the computational cost of traversing through the pairs of convex elements outweighs that of the simple feed-forward prediction in LOCC.

Separate Axis Theorem (SAT) is another widely used method for detecting collisions between convex objects. It works by projecting the shapes onto different axes and checking for overlap in all dimensions. If there is no overlap on any axis, then the objects do not collide. This method can be efficiently implemented on GPUs, as each axis can be processed in parallel and the results can be combined to determine if a collision has occurred. SAT requires only simple arithmetic operations, such as dot products and comparisons, which can be easily accelerated by GPUs and is used in Brax. Despite its efficiency, SAT has limited scaling capability when dealing with complex non-convex shapes because it requires checking over $N_1 N_2$ axes, where N_1 and N_2 are the numbers of edges for objects 1 and 2 respectively.

III. Local object crop collision network

We now describe LOCC which directly evaluates the collision between two objects using their meshes and poses.

[1]If objects are sufficiently faraway, you can rule them out in the Broad phase of collision detection.

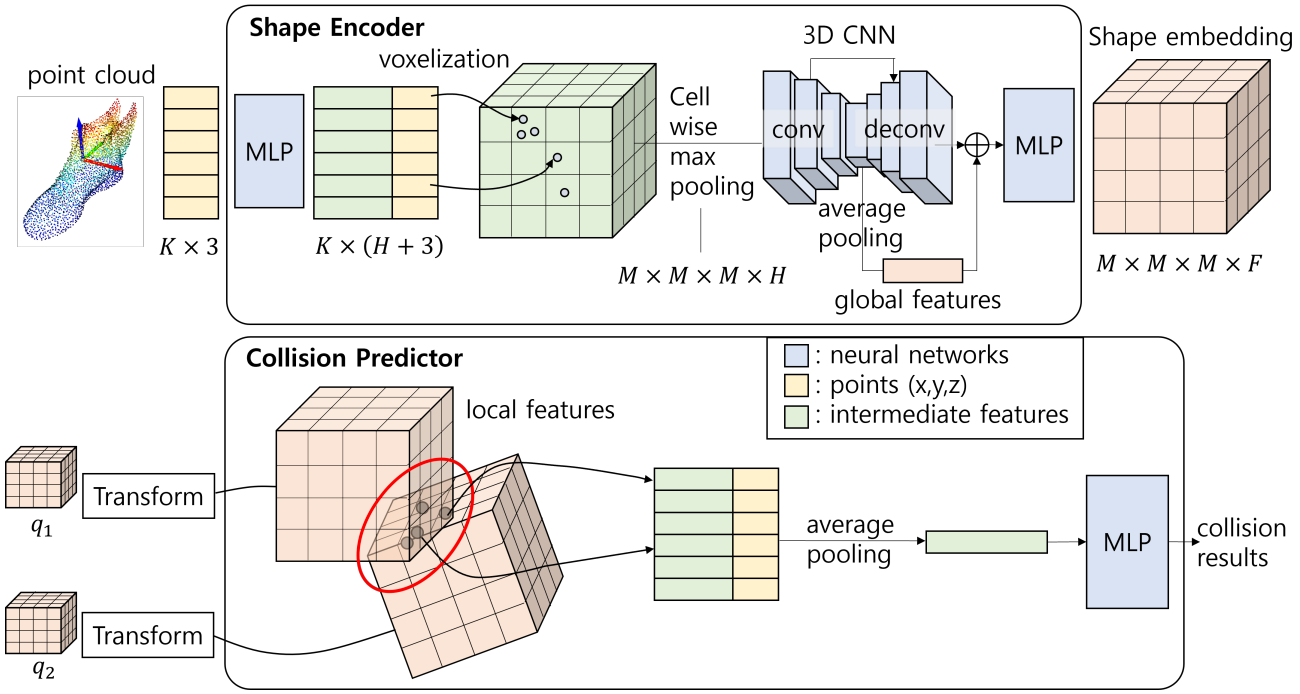


Fig. 4: Computational flow in LOCC. In LOCC, there are two modules: shape encoder, and collision predictor. The shape encoder takes an object shape represented in point cloud as an input, and generates a shape embedding. The collision predictor takes two shape embeddings and object poses as inputs, and outputs a collision result. K denotes the number of points in the point cloud of an object shape, M denotes the size of the voxel grid defined on the AABB of an object shape, and H and F denote the output dimensions of the first and second MLPs in the shape encoder respectively. Cross-circle symbol in the shape encoder indicates concatenation.

There are two modules in LOCC: shape encoder and collision predictor. Like SceneCollisionNet [6], our shape encoder uses a voxel grid defined on the object point cloud and computes cell-wise features. Unlike SceneCollisionNet however, we exploit locality by passing to the collision predictor only the local crops of the shape embeddings that are in a collision to facilitate data efficiency. We now walk through Figure 4 that describes the computations in LOCC.

We first create a point cloud by sampling K number of points using the mesh of an object. The shape encoder begins its computation by processing each point using a multilayer perceptron (MLP), which outputs $K \times H$ features, where H is the output dimension of the MLP. We then compute the AABB of each object's mesh and define a voxel grid of size $M \times M \times M$ over the AABB. We put the features into the voxel grid according to the coordinates of their corresponding points to get features of size $M \times M \times M \times H$. We process the feature for each cell using cell-wise max-pooling over the features. We further process the local feature for each cell by applying a 3D Convolutional Neural Networks (CNNs) with skip connections to generate the shape embedding of size $M \times M \times M \times F$, where F is the dimension of the feature at each cell.

In the 3D CNN, we use a U-Net-like architecture [23] but add an average pooling operation just before the deconvolution to extract a global feature. This global feature is then broadcasted to the local features of each cell just after the deconvolution. The intuition is to encode both local and global features into each cell feature.

For the collision predictor, we first create the OBB of shape embeddings of two objects by using their poses. Then, we select the features in colliding cells. One way to obtain the cells in a collision is by considering the center point of each cell of the OBB, and checking if that point is inside of another object's OBB. The downside of this approach is that while the center point may be outside the OBB, the cells may still be in a collision. To solve this, we add a margin to each cell of OBB whose magnitude is the distance from the center point to a vertex of a cell. This way, we are guaranteed not to have false negatives in detecting the cells in a collision. Using the features of the selected cells, we apply average pooling, and the resulting vector gets passed to a collision predictor which then makes a prediction.

Note that while margin-padding guarantees no false negative in collision detection, it may result in false positives, and include more features than needed. However, we found that adding more information does not usually hurt the performance of collision prediction.

There are some notable hyperparameters that are worth mentioning. First, M is a particularly important hyperparameter and should be chosen based on object sizes. If M is too small, then the shape embedding will not contain sufficient information. If M is too big, then it will take up a lot of

memory and checking collisions between two OBBs would take a long time. After testing values of 5, 6, and 7, we found that 6 provides a good balance of accuracy and efficiency.

For K, which is the number of points fed into the shape encoder, its value doesn't impact testing efficiency because we keep the shape embedding at a fixed size through cell-wise max pooling, even when K changes. We set K to 1500 so that we can generate enough points to cover the surface and capture shape details.

A. Dataset preparation and training

To prepare the dataset we use existing object datasets such as YCB or Google ScanNet [4, 9]. Our dataset has the form $\{(x_1^{(i)}, x_2^{(i)}, q_1^{(i)}, q_2^{(i)}, y^{(i)})\}_{i=1}^n$ where x_1 and x_2 denote object meshes, q_1 and q_2 denote object poses, and y denotes a collision label. This data is generated by synthetically creating a collision between two objects.

A naive approach to generate such a dataset is to first sample two objects from an object set, place them at poses uniformly sampled from a pre-defined bound, and evaluate the collision to assign y. However, we found that uniform random sampling mostly generates trivial cases where two objects are either too far away or overlap severely. This would make LOCC fragile to non-trivial cases where objects overlap only slightly.

So, we devised a new strategy: after we sample poses with large enough bounds, we manipulate the distance between two objects to create collisions with different overlapping volumes. Figure 5 demonstrates our strategy. More concretely, we first compute the shortest-distance vector, δ, between two objects, and then translate one of them along the direction of δ. This way, for a given object pose pair, we can get collision data points at different overlapping volumes by manipulating the magnitude of δ. In our implementation, we sample the target magnitude $|\delta'|$ from a normal distribution with zero mean and 0.020m standard deviation, and take an absolute value.

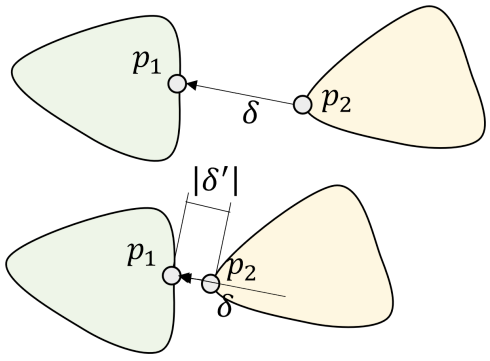


Fig. 5: An illustration of distance manipulation for generating the collision dataset. We first place two objects (green and yellow) far apart by sampling their poses. We then compute the minimum distance vector, δ, and points p_1 and p_2 on this vector on each object. Then, we move one object by $|\delta| - |\delta'|$ in the direction of δ to obtain a data point with the desired distance of $|\delta'|$. The desired distance $|\delta'|$ is sampled from a normal distribution.

Using this dataset, LOCC is trained end-to-end with binary cross-entropy loss, with a regularization term to prevent the shape encoder from overfitting. The loss function of LOCC is given by

$$\sum_{(d,y)\in\mathcal{D}} BCE(f^{CP}(f^{SE}(d)), y) + \alpha|f^{SE}(d)|^2 \ ,$$

where $\mathcal{D} = \{d^{(i)}, y^{(i)}\}_{i=1}^n$ is dataset, $d^{(i)} = \{x_1^{(i)}, x_2^{(i)}, q_1^{(i)}, q_2^{(i)}\}$ is the input, f^{SE} is the shape encoder, f^{CP} is the collision predictor, and $BCE(\hat{y}, y)$ is the binary cross entropy between the prediction and label y.

During training, we apply a data augmentation scheme to make LOCC robust to different orientations. Concretely, suppose we have a data point $\{x_1^{(i)}, x_2^{(i)}, q_1^{(i)}, q_2^{(i)}, y^{(i)}\}$. We first randomly sample a rotation matrix R and then apply R to both shapes, and R^{-1} to both poses to make a new data point $\{R \cdot x_1^{(i)}, R \cdot x_2^{(i)}, R^{-1} \circ q_1^{(i)}, R^{-1} \circ q_2^{(i)}, y^{(i)}\}$, where $R \cdot x$ denotes rotating the shape by R, and $R \circ q$ is applying rotation R to the rotational part of pose q. Note that we can keep the same label even after these operations because $\{x, q\}$ and $\{R \cdot x, R^{-1} \circ q\}$ occupy the same volume in the 3D space.

IV. Experiments

A. Details of training

For the hyperparameters of LOCC , we set K to 1500, and uniformly sample 1500 points from the surface of the object mesh. In the shape encoding, we have $M = 6, H = 256$, $F = 64$. More details about the architecture and the training hyperperameters are included in the supplementary material.

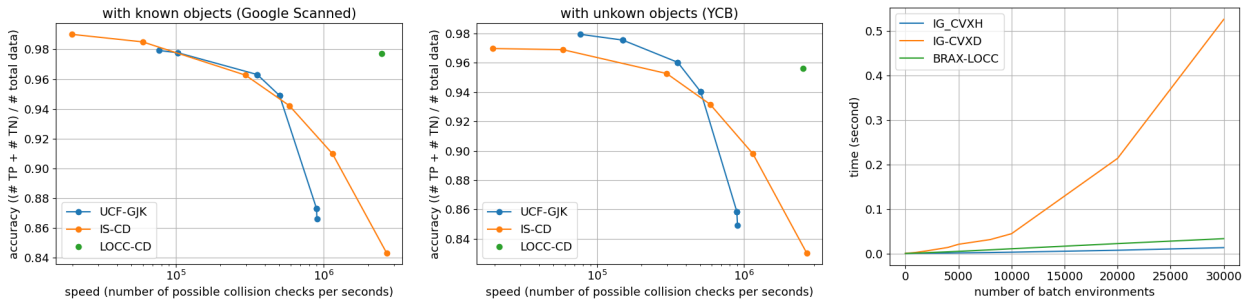
We use Google Scanned Object (GSO) [9] as our data that has 1030 object meshes most of which are non-convex. We use the meshes to generate point clouds and define the center of the mesh as the center of AABB. Unless otherwise mentioned, we generate a total of 230 million data points with 230K object pairs with the strategy in Sec. III-A. For training, the learning rate is set to 0.001 and the batch size to 32. The Adam [15] optimizer is used with α set to 0.5 in the loss. See the appendix for more information on network parameters.

For testing, we use two distinct test sets. The first test set, called *known object set*, uses the same set of objects as the training set, but different poses and object pairs. The total number of object pairs in the training set is 230K, and there are approximately 1 million possible object pairs. The second test set, called *unknown object set*, consists of 30 novel objects chosen from YCB object set [4] that were not used during training. In generating the poses for test sets, we use uniform sampling rather than the method from section III-A to better characterize the poses that will be encountered during simulation.

B. Results

We wish to validate the following claims through our experiments:

- *Claim 1 (computational efficiency of* LOCC*)* For non-convex objects, CD with LOCC is more computationally



(a) (b) (c)

Fig. 6: Accuracy-speed plots for (a) known and (b) unknown object sets. # TP stands for number of true positive data pairs, and # TN stands for number of true negative data pairs. (c) Plot of computational time it takes to simulate Δt vs number of environments being simulated in parallel.

efficient than analytical CD algorithms such as GJK and CD based on implicit shape representation such as [10].

- *Claim 2 (data efficiency)* LOCC is more data efficient than the standard OCN that learns the global object shapes to determine collision such as [6].
- *Claim 3 (computational efficiency and generality in physics simulation)* For non-convex objects, physics simulation with BRAX-LOCC, which integrates Brax, LOCC, and the contact resolution algorithm from [24], is faster and has higher fidelity than IsaacGym, and more general than Brax which can only simulate convex objects.

To support *Claim 1 (computational efficiency)*, we use two baselines. The first is the UCF version of GJK written in JAX so that it can utilize XLA to run faster than the standard GJK on a GPU. We denote this as UCF-GJK. The second is CD based on implicit surface (IS), denoted IS-CD, which determines collision based on a set of query points as in [10].

Unlike LOCC whose online computation time is fixed and its accuracy depends on the off-line dataset, UCF-GJK and IS-CD must trade off their computation time for higher accuracy by increasing the number of elements in decomposition or the number of query points. Therefore, we use the accuracy versus speed plot to evaluate their performances, where the speed is defined as the number of possible collision checks in a second, and accuracy is defined as the number of true positive and true negative predictions divided by the total number of predictions. For UCF-GJK, there are multiple hyperparameters, so we tested several hyperparameters and report the one that attains the best performance.

Figures 6a and 6b show the results for the known and unknown object test sets and respectively. In Figure 6a, the result shows that LOCC achieves the accuracy of about 98% and 96% for known and unknown object sets respectively. This is comparable to the best accuracy attained by UCF-GJK and IS-CD, which are 98.5% and 98% for known and unknown object sets respectively. However, we can see that LOCC achieves its best accuracy at a speed at least 10 times faster than both of these methods. This illustrates the advantage of LOCC: by directly predicting collisions using a NN, it attains a faster computation time than UCF-GJK and IS-CDwith comparable accuracy.

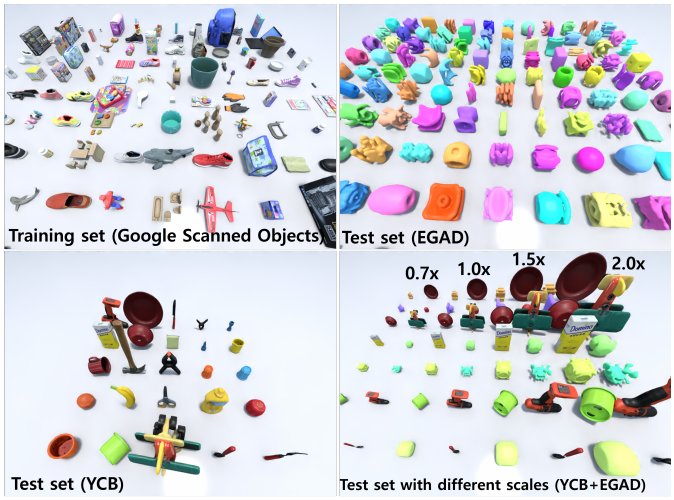


Fig. 7: This figure illustrates a selection of objects utilized during training and testing phases. The upper left quadrant features Google Scanned Objects [9] employed for the training process. During testing, objects from both the YCB [4] (bottom left) and EGAD [20] (upper right) datasets are implemented to demonstrate the generalization capability of our LOCCmodel. Further, we assess LOCC's adaptability to varying object scales by adjusting the YCB and EGAD objects by factors of 0.7x, 1.0x, 1.5x, and 2.0x (displayed in the bottom right quadrant). The result is a consistent collision accuracy exceeding 95% across all test cases.

Pursuing our investigation into the generalization capability of our method, we conducted supplementary experiments that involved training and testing datasets with significantly varied shapes. In these experiments, our model was trained solely on the GSO dataset, and then tested on two distinct datasets: EGAD [20], a compilation of unusually-shaped objects created to evaluate the robustness of grasping algorithms, and the YCB dataset [4], which we adjusted for scale (0.7x, 1.0x,

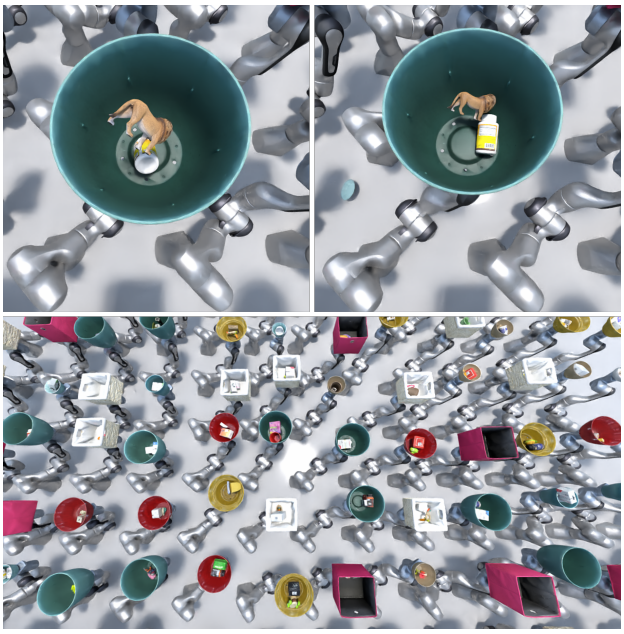


Fig. 8: Object shaking simulation task comparing IsaacGym and BRAX-LOCC.

Methods	# objects			
	5	20	50	100
GLOBAL-OCN	0.67±0.02	0.83±0.03	0.92±0.02	0.93±0.01
LOCC	0.77±0.02	0.90±0.02	0.93±0.01	0.93±0.01

TABLE I: Accuracy with varying number of objects in the training data.

Fig. 9: Example of inaccurate simulation behavior in IG-CVXH due to a convex hull approximation of objects. The objects should fall into the bowls but are "floating" in the air because the bowls have been convexified

1.5x, and 2.0x). Representative samples from both the training and testing stages can be found in figure 7. Remarkably, our method consistently delivered prediction accuracy of at least 95% across all the testing datasets. Such results substantiate the robust adaptability of LOCC.

To support *Claim 2 (data efficiency)*, we adapt SceneCollisionNet [7] to object-object collision detection. This method does locality and must learn object shapes. We denote this method as GLOBAL-OCN. The shape encoder of GLOBAL-OCN is the same as LOCC, except that the features are globally average-pooled before the de-convolution network in the collision predictor to extract the global features. To check the data efficiency, we train GLOBAL-OCN and LOCC with a varying number of object shapes, and for each number of objects, we collect 0.1 million object pairs and poses and test LOCC and GLOBAL-OCN on the novel object set. Table I shows the result.

When the number of training objects is 5 or 20, LOCC has 7-10% higher accuracy than GLOBAL-OCN on average. As you increase the number of training objects, the gap gets smaller, indicating that although GLOBAL-OCN can eventually learn to generalize to novel objects, LOCC does so with a smaller number of object shapes. This supports our intuition shown in Figure 3: LOCC outperforms GLOBAL-OCN because while LOCC can just encode the local geometric patterns at contacts which are similar across different objects, GLOBAL-OCN must learn to encode global object shapes for which there is much more variation.

Finally, to validate *Claim 3 (computational efficiency in physics simulation)*, we compare BRAX-LOCC with IsaacGym on non-convex object simulation. We simulate 200 distinct environments with 3 objects, one of which is a bowl, and the rest are randomly selected from the Google Scanned Object set. The objects get dropped into the bowl, and the bowls get continuously shaken to create contacts. The example scenes are shown in Figure 8, and its video is included in the supplementary material. We could not compare BRAX-LOCC to Brax because it is limited to convex shapes. We set the size of the simulation time step, Δt, to 0.01/4 seconds, with 4 sub-steps.

We measure the simulation speed without rendering. As Figure 6c shows, BRAX-LOCC is more than 10 times faster than IG-CVXD, which uses GJK and V-HACD, when we are simulating 30,000 environments. BRAX-LOCC is slightly slower than IG-CVXH, which uses a convex hull approximation of the object shapes. However, IG-CVXH has significant issues in simulating non-convex shapes, as demonstrated in Figure 9.

To measure this quantitatively, we compute the absolute sign distance among all the objects whenever a contact arises, and take the minimum value. We denote this as min-|sd|. This quantity measures the penetration depth if the objects penetrate each other at the contact or the distance between two objects when the contact is falsely detected. For an ideal simulator, this value would be zero at every contact. We use default parameters for GJK in IsaacGym. The results are shown in Table II.

As the table indicates, BRAX-LOCC has the min-|sd| value closest to zero among all the baselines, indicating that it has the least penetration depth and the least number of false positives. This is because IG-CVXD and IG-CVXH inevitably lose their accuracy due to shape approximation, while the accuracy of LOCC depends solely on the quality and quantity of the offline training data.

V. Conclusion

We proposed a novel OCN, LOCC, that compared to previous approaches [8, 6], is more data or computationally effi-

Simulator	metric (unit: meter)		
	average	top-10% average	maximum
IG-CVXD	0.0132	0.0302	0.0392
IG-CVXH	0.0709	0.1266	0.1286
BRAX-LOCC(ours)	0.0077	0.0218	0.0338

TABLE II: Statistics on 600 absolute signed distance between two objects. *average* is the average of min-|sd|, *top-10% average* is the average of highest 10%, and *maximum* is the highest min-|sd| out of 600.

cient. We showed that compared to analytical contact detection algorithms, our approach achieves higher accuracy and speed when it comes to simulating non-convex objects in multi-environment scenarios by making better use of GPU resources. We integrated LOCC into the open-source physics engine, Brax, along with the contact resolution algorithm from [24] and showed that BRAX-LOCC simulates non-convex objects while Brax cannot, and show that it outperforms IsaacGym in terms of speed and fidelity.

REFERENCES

[1] Speeding up reinforcement learning with a new physics simulation engine. https://ai.googleblog.com/2021/07/speeding-up-reinforcement-learning-with.html.

[2] XLA: Optimizing compiler for machine learning. https://www.tensorflow.org/xla.

[3] Mihai Anitescu and Florian A Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997.

[4] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The YCB object and model set: Towards common benchmarks for manipulation research. In *International Conference on Advanced Robotics (ICAR)*, pages 510–517. IEEE, 2015.

[5] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In *European Conference on Computer Vision*, pages 608–625. Springer, 2020.

[6] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6010–6017. IEEE, 2021.

[7] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. *International Conference on Robotics and Automation*, 2021.

[8] Nikhil Das and Michael Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114, 2020.

[9] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3D scanned household items. *arXiv preprint arXiv:2204.11918*, 2022.

[10] Danny Driess, Jung-Su Ha, Marc Toussaint, and Russ Tedrake. Learning models as functionals of signed-distance fields for manipulation planning. In *Conference on Robot Learning*, pages 245–255. PMLR, 2022.

[11] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[12] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.

[13] Sami Haddadin, Alessandro De Luca, and Alin Albu-Schäffer. Robot collisions: A survey on detection, isolation, and identification. *IEEE Transaction on Robotics*, 2018.

[14] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3D scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac Gym: High performance GPU-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[17] Khaled Mammou. V-hacd. https://github.com/kmammou/v-hacd.

[18] Khaled Mamou, E Lengyel, and A Peters. Volumetric hierarchical approximate convex decomposition. In *Game Engine Gems 3*, pages 141–158. AK Peters, 2016.

[19] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 289–298, 1988.

[20] Douglas Morrison, Peter Corke, and Jürgen Leitner. Egad! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. *IEEE Robotics and Automation Letters*, 5(3):4368–4375, 2020.

[21] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

[22] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on*

computer vision and pattern recognition, pages 165–174, 2019.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[24] Dongwon Son, Hyunsoo Yang, and Dongjun Lee. Sim-to-real transfer of bolting tasks with tight tolerance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9056–9063. IEEE, 2020.

[25] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11358–11367, 2021.

[26] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.008.