An Automated Real2Sim System for Dynamically Accurate Simulation Assets

Nicholas Pfaff*†, Lirui Wang* and Russ Tedrake*
*Massachusetts Institute of Technology †Imperial College London
https://nepfaff.github.io/object_real2sim/

***Abstract*—Training robots in simulation and executing the learned skills in the real world has become a popular framework. Yet, doing so requires practitioners to construct a dynamically accurate digital replica of the real-world scene, which is time-consuming and requires domain expertise. We propose an automated Real2Sim pipeline for rapidly creating such simulation assets for real-world objects. In contrast to previous work, we focus on achieving good simulation performance in highly dynamic tasks, i.e., tasks in which the object is subject to high accelerations and varying contact interactions. Through extensive simulation experiments, we show that our system is capable of producing simulation assets that are both dynamically accurate and fast to simulate.**

I. Introduction

Physics simulation has fueled many recent advances in robotics by enabling learning skills in simulation and executing them in the real world [1]–[5]. This approach, termed Sim2Real, is desirable as many machine learning-based approaches require large quantities of interaction data that are much easier to obtain in simulation than in the real world. An example of such a learning-based approach is reinforcement learning [1], [2]. The Sim2Real approach usually requires researchers to construct a replica of the real-world scene in which the robots will be trained. Building such a replica involves manually curating object geometries and tuning their dynamic parameters, such as mass and moment of inertia. This manual process requires domain expertise and is time-consuming. Consequently, it is hard to scale this process to a large variety of real-world scenes, creating a Sim2Real gap. A Sim2Real gap refers to the difference between the simulation environments in which the robots are trained and the real-world environments in which the robots are tested. Skills learned in simulation will not work in the real world if the Sim2Real gap is too big.

We propose an automated pipeline for rapidly creating dynamically accurate simulation assets for real-world objects. Our work is inspired by recent works that have attempted to automate the scene replication process and termed the problem *Real2Sim* [6]–[10]. Some of these works focus on identifying the dynamic parameters of existing models [6], [7], while others focus on reconstructing scene geometry [8], [9]. In this work, we focus on both simultaneously by automatically constructing a Unified Robot Description Format (URDF) for an object of interest. A URDF is a simulation asset description format supported by most state-of-the-art physics simulators [11]–[13]. It specifies the object's appearance/ visual geometry, the collision geometry used for collision checking during simulation, and the physical properties such as mass and moment of inertia. Our approach differs from previous work in that we focus on automatically creating high-quality and dynamically accurate simulation assets of objects of interest. By dynamically accurate, we mean accurate simulation performance when the object's accelerations cannot be ignored. Dynamic scenarios contrast with quasi-static scenarios, where the object's accelerations can be considered insignificant and thus ignored. Most existing works focus on these quasi-static scenarios, such as object pick-and-place tasks, where the object's dynamics become insignificant while the object is being grasped. We instead focus on highly dynamic tasks such as pushing and rolling.

Our pipeline consists of a robotic system for automatically collecting multi-view RGB images of a real-world object. We then mask out the object of interest using a segmentation network and use these masked images to train a state-of-the-art geometric reconstruction network. To obtain the visual geometry, we extract a mesh representing geometry as a set of vertices and faces from the reconstruction network's representation. To obtain the collision geometry, we propose to approximate this mesh with primitive shapes, such as boxes and cylinders. We obtain the object's physical properties by measuring its mass and computing the center of mass and moment of inertia using a uniform density assumption.

We evaluate our pipeline both qualitatively and quantitatively on pushing and rolling tasks. While doing so, we attempt to answer what makes a suitable collision geometry for fast and dynamically accurate simulations by comparing our various primitive approximations with an existing state-of-the-art geometric simplification method [14]. Our results demonstrate that our pipeline can produce dynamically accurate simulation assets that achieve low simulation errors even for highly dynamic scenarios such as an object rolling down a ramp. We also show that we can find an entire collection of collision geometries that achieve different simulation performance tradeoffs. This is useful as it enables choosing a collision geometry based on the desired simulation properties, which are downstream application dependent. Specifically, we examine the tradeoff between dynamical accuracy, simulation time, and contact point smoothness/ continuity. We show that we can find primitive-based collision geometries that outperform the baseline in all of these metrics. Our collision geometries

surpass the baseline by more than 47% in terms of simulation accuracy and more than 73% in terms of contact point continuity. We additionally give recommendations for choosing the best collision geometry based on downstream application requirements, enabling users to adapt our pipeline performance based on their needs.

In summary, our key contributions are:

- A practical Real2Sim pipeline for automatically creating high-quality and dynamically accurate simulation assets for real-world objects.
- A novel method for finding a collision geometry composed of primitive shapes.
- An evaluation of the tradeoffs between dynamical accuracy, simulation time, and contact point continuity for a large set of automatically constructed collision geometries.

II. Related Work

A. 3D Reconstruction

3D reconstruction methods focus on reconstructing geometry from real-world observations such as RGB and depth images. These methods produce either implicit [15]–[18] or explicit [19]–[21] representations. Explicit methods represent geometry directly, such as with a triangle mesh. A triangle mesh consists of a set of vertices in $\mathbb{R}^3$ that are connected by triangular faces. Implicit methods indirectly represent geometry using a continuous function and consequently can represent geometry in greater detail. Recently, there was a breakthrough in learning such implicit functions from only RGB images, inspired by the seminal work NeRF [16]. We use a state-of-the-art version of NeRF as provided by the open-source library Nerfstudio [22].

B. Real2Sim

Real2Sim is the problem of automatically generating a digital replica of a real-world scene to reduce the simulation's Sim2Real gap. Most geometric Real2Sim approaches focus on reconstructing the geometry of the entire scene while treating the scene as one rigid object. Consequently, the individual objects in the scene cannot be separated from their supporting surfaces, which makes this approach unsuitable for simulating robotic manipulation tasks such as pick-and-place. A representative algorithm is [23]. Le Cleac'h et al. [10] propose a Real2Sim system for reconstructing an object's geometry from observations and identifying its dynamic parameters, such as mass and moment of inertia. They first obtain an implicit model of the object's geometry from RGB images and then propose a novel simulation approach for directly simulating their implicit model. Their method achieves dynamically accurate object-level Real2Sim from only RGB observations. However, their proposed physics simulation approach is incompatible with the state-of-the-art simulators used for training robots in simulation [11]–[13], excluding their approach from being used in current robotics systems. Wang et al. [8] propose a Real2Sim system for constructing explicit object mesh representations from depth images and using these representations to train robotic manipulators on grasping tasks. Their object-centric mesh representations fit directly into state-of-the-art simulators, making them suitable for training robots in simulation. However, they focus on simple picking tasks with little dynamics. We instead propose to use state-of-the-art reconstruction methods to reconstruct object-centric meshes from only RGB images and simplify these meshes to create collision geometries that enable both fast and dynamically accurate simulations.

C. Geometric Mesh Simplification

Geometric mesh simplification is the process of simplifying triangle meshes by reducing the number of vertices and faces while maintaining as much fidelity as possible. One family of work tackles this problem by iteratively reducing the number of vertices [24], [25]. A different approach to mesh simplification is convex decomposition. Convex decomposition decomposes a triangle mesh into a set of almost convex components and simplifies these individual components by taking their convex hull [14], [26]. The union of these parts is then taken as the simplified mesh. Another series of works approximates meshes using primitive shapes, such as spheres [27]–[29].

Simplifying triangle meshes before using them as collision geometries in physics simulators is common practice. The standard simplification method for this purpose is convex decomposition, which is also the approach taken by Wang et al. in their Real2Sim pipeline [8]. Instead, we propose using primitive shape approximations for our collision geometries as we found this method more suitable for simulation.

III. Problem Statement

Given a sequence of RGB images, camera poses, segmentation masks, and a mass estimate of the object of interest, our pipeline constructs a simulation asset of the object. This asset consists of a URDF, containing a collision geometry $\mathcal{C}$, a visual geometry $\mathcal{V}$, and physical properties $\mathcal{P}$. We additionally propose automatically collecting these inputs using a robotic manipulator with a wrist-mounted camera, reducing the required input to the actual object $\mathcal{O}$ and its mass measurement m.

IV. Using a Robot System to Automatically Create Simulation Assets from Real-World Objects

We provide descriptions of the robotic system in subsection IV-A, the geometric reconstruction framework in subsection IV-B, the geometric simplification approach in subsection IV-C, the physical property estimation method in subsection IV-D, and the URDF construction in subsection IV-E. The system diagram is shown in Fig. 1.

A. Robotic system for data collection

Our pipeline requires multi-view posed RGB images of the object of interest, where camera poses refer to the translation and orientation of the camera in the world. Our

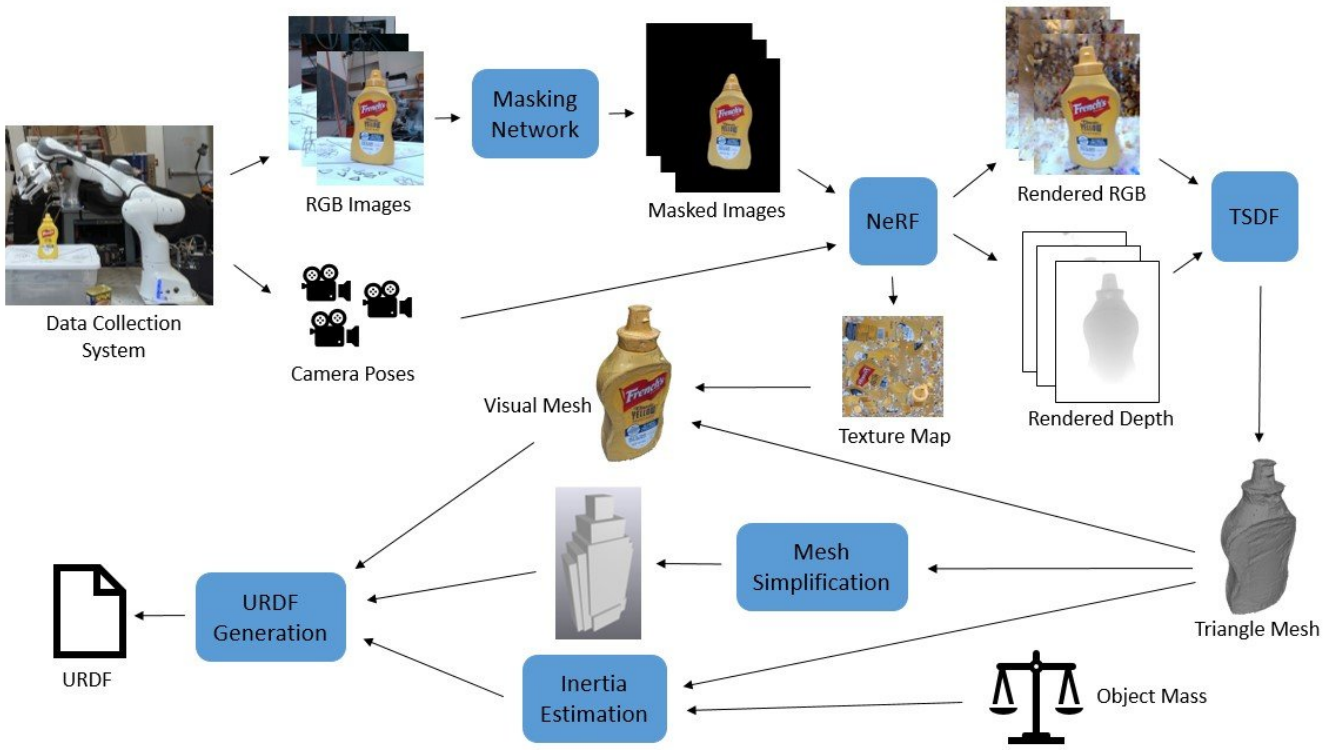

Fig. 1: An overview of the system pipeline. The arrows indicate the pipeline flow.

reconstruction, especially the visual reconstruction, performs better with denser object views. Manually recording hundreds of images and measuring their associated camera poses is time-consuming and, consequently, does not scale well. To mitigate this issue, we propose to use a robotic system to collect the required data automatically. In particular, we use a *Panda Emika Franka* robot with a wrist-mounted *Intel RealSense D435* camera. We determine an object location in the center of the robot's workspace, i.e., the region reachable by the robot arm, and manually determine robot poses that lead to dense views of that location. At data collection time, we place the object of interest at the object location and command the robot to the pre-determined locations using the MoveIt motion planning library [30]. We determine the camera poses by obtaining the wrist-camera pose relative to the robot pose through MoveIt's camera calibration library. Camera calibration is approximate, which leads to inconsistent errors in the recorded camera poses. We use COLMAP's structure from motion pipeline [31] to refine these inconsistent camera poses. More precisely, we initialize COLMAP with the camera poses from the wrist-camera calibration, refine the poses using COLMAP, and then re-align the COLMAP poses with the wrist-camera poses using COLMAP's model aligner. Inconsistent lighting can drastically reduce the image quality. We found that ensuring consistent lighting is critical for achieving good reconstruction results. We achieve this by mounting a commonly available light ring above the object location. Fig. 2 shows our robot setup.

Fig. 2: The robot system for automatic data collection. It consists of a *Panda Emika Franka* robot with a wrist-mounted *Intel RealSense D435* camera and a light ring.

1) Masking: We only want to reconstruct the object of interest $\mathcal{O}$ instead of the entire scene. We achieve this by masking out the object in our collected images and only using the masked object pixels for our reconstruction while ignoring the background pixels. We obtain the object masks using a Mask R-CNN segmentation network [32]. Mask R-CNN predicts the probability of each image pixel belonging to the object of interest. We keep all pixels that have a probability greater than 50%.

B. Geometric reconstruction

We use neural radiance fields (NeRF) for geometric reconstruction as NeRF enables accurate geometric reconstruction from only RGB images.

1) Neural Radiance Fields: Concretely, NeRF is a continuous function $F_{\boldsymbol{\theta}} : (\boldsymbol{x}, \boldsymbol{d}) \mapsto (\boldsymbol{c}, \sigma)$ that maps a 3D position $\boldsymbol{x} \in \mathbb{R}^3$ and a 3D viewing direction $\boldsymbol{d} \in \mathbb{R}^3$, $||\boldsymbol{d}|| = 1$ to the corresponding view-dependent color $\boldsymbol{c} \in [0, 1]^3$ and volume density $\sigma \in [0, \inf)$. $\boldsymbol{\theta}$ are learnable parameters. σ represents the probability of hitting a particle while traveling an infinitesimal distance. We can use differential volume rendering to render out an image $\boldsymbol{I}$ at a desired camera pose. This is achieved by marching a camera ray $\boldsymbol{r}(t) = \boldsymbol{x_o} + t\boldsymbol{d}$ through each pixel, where $\boldsymbol{x_o}$ is the camera origin. The RGB color of the image is then calculated as

$$\boldsymbol{I_\theta}(\boldsymbol{r}) = \int_{t_n}^{t_f} w(t)\boldsymbol{c_\theta}(\boldsymbol{r}(t), \boldsymbol{d})\, dt, \tag{1}$$

where t_n, t_f denote the near and far bounds of the ray and

$$w(t) = \exp(-\int_{t_n}^{t_f} \sigma_{\boldsymbol{\theta}}(\boldsymbol{r}(s))\, ds)\sigma_{\boldsymbol{\theta}}(\boldsymbol{r}(t)). \tag{2}$$

$w(t)$ describes the likelihood of a ray terminating at t. The parameters $\boldsymbol{\theta}$ can then be optimized using gradient descent and a mean squared error loss:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{I}\in\mathcal{D}, \boldsymbol{r}\in\mathcal{R}(\boldsymbol{I})}||\boldsymbol{I}(\boldsymbol{r}) - \boldsymbol{I_\theta}(\boldsymbol{r})||_2^2 \tag{3}$$

where $\mathcal{D}$ is the set of training images and $\mathcal{R}(\boldsymbol{I})$ is the set of rays through each pixel of $\boldsymbol{I}$. Using the ray termination likelihood $w(t)$, it is additionally possible to render out the expected depth image:

$$\boldsymbol{D_\theta}(\boldsymbol{r}) = \int_{t_n}^{t_f} w(t)||\boldsymbol{x_o} - t\boldsymbol{d}||_2\, dt. \tag{4}$$

For the sake of fast training and state-of-the-art performance, we use Nerfstudio's *Nerfacto* method for $F_{\boldsymbol{\theta}}$ [22].

2) Extracting a Mesh from NeRF: NeRF represents geometry implicitly using a function $F_{\boldsymbol{\theta}} : (\boldsymbol{x}, \boldsymbol{d}) \mapsto (\boldsymbol{c}, \sigma)$, where the volume density σ can be interpreted as the physical density. However, most physics simulators expect an explicit geometry, such as a triangle mesh [11]–[13]. Extracting a triangle mesh from NeRF requires us to find a discrete approximation of the continuous density. Two popular ways of achieving this are running marching cubes [33] on uniform density grid samples from NeRF and TSDF (truncated signed distance function) fusion [34].

We picked TSDF fusion as it results in locally smoother meshes which is beneficial for mesh texturing (see section IV-B3). TSDF fusion uses RGB and depth images from different camera poses to obtain a TSDF. A triangle mesh can be obtained from this TSDF using marching cubes with the TSDF's zero-level set. We use NeRF to render a sequence of RGB and depth images, using a camera trajectory that we obtain by interpolating between the camera poses in the training set. We then use TSDF fusion to obtain a mesh from these RGB and depth images, as implemented in Nerfstudio.

The raw mesh $\mathcal{M}$ produced by this pipeline differs from the real-world object $\mathcal{O}$ in both scale and pose as Nerfstdio normalizes the camera poses before training a NeRF. We apply the inverse of this normalization operation to recover the real-world scale and pose. We then normalize the mesh's pose by translating it to the world origin and rotating it such that its longest side is aligned with the world's z-axis and the shortest side with the world's x-axis. In particular, we consider the mesh's longest side to be the principal component and the shortest side the minor component of the mesh's vertices, as obtained from principle component analysis (PCA). We perform this pose normalization to achieve a canonical pose for all our simulation assets.

3) Texturing a Mesh with NeRF: A raw triangle mesh only contains geometric but no visual information. We add visual information through texture mapping, i.e., applying an image to the surface of the triangle mesh $\mathcal{M}$ through coordinate mapping. We use Nerfstudio for creating this texture mapping. Nerfstudio first uses the xatlas library [35] to unwrap the mesh onto a texture image, resulting in each mesh face having an associated location on the texture image. It then uses NeRF to generate the texture image pixel values. More precisely, for every pixel in the texture image, it finds the corresponding 3D mesh position and normal vector. It then renders a ray along this normal vector with NeRF to obtain the texture image RGB value, as described in section IV-B1.

C. Geometric simplification

The raw TSDF mesh $\mathcal{M}$ from NeRF consists of thousands of vertices and faces, leading to slow simulation times, i.e., simulation takes significantly longer than real-time. For practical applications, we desire our simulation assets to enable faster than real-time simulations while providing high dynamic accuracy. Inspired by the fact that many objects are composed of basic primitive shapes for manufacturing reasons, we propose to approximate the raw mesh using primitive shapes. Firstly, we resolve ambiguity at the object's bottom. Secondly, we sample signed distance values. Lastly, we fit primitives to the mesh with a loss based on the sampled signed distance values.

1) Resolving Object Bottom Ambiguity: Our pipeline input consists of an object $\mathcal{O}$ standing on an opaque surface, as shown in Fig. IV-A. Our collected data thus does not contain any views of the object's bottom. As a consequence of this ambiguity, the NeRF mapping $F_{\boldsymbol{\theta}} : (\boldsymbol{x}, \boldsymbol{d}) \mapsto (\boldsymbol{c}, \sigma)$ results in arbitrary density and color values at 3D positions $\boldsymbol{x}$ at the

object's bottom. This leads to the extracted mesh $\mathcal{M}$ having arbitrary triangles and faces at the object's bottom. We assume that the object's bottom is flat and hence crop the mesh at the standing surface height by removing all triangles and faces below this height. We know this height as we know the actual pose of the mesh in the world from the inverse normalization operation. After cropping, the mesh has a hole in the bottom. We close this hole by computing the mesh union between the cropped mesh and a new mesh bottom that we obtain using convex decomposition. In particular, we obtain the bottom by computing the convex decomposition of the cropped mesh with V-HACD [26] and cropping this convex decomposition just above the standing surface height. This cropped convex decomposition results in a flat surface that can be used as the new object floor. We denote the cropped and closed mesh as $\mathcal{M}_{closed}$.

2) Signed Distance Sampling: After resolving the mesh bottom, we scale the mesh to fit inside a sphere with a radius of one meter (unit sphere). We then sample 600 thousand signed distance values around the mesh surface and 500 values uniformly in a sphere of radius 1.3 meters. The signed distance is the distance from the sample point to the mesh surface, where points inside the mesh have a negative distance and points outside the mesh have a positive distance. We use a ray stabbing method [36] to determine the signed distance values as it is robust to non-watertight meshes and meshes with internal structures. This robustness is essential as our NeRF meshes $\mathcal{M}_{closed}$ are not watertight and have internal structures. Non-watertight means that the mesh surface has holes, while internal structures imply that triangles are also present inside the mesh rather than just at the mesh's surface. We use the CUDA kernel implementation of this method from Hao et al. [28] to achieve fast sampling performance.

3) Primitive Fitting: Our primitive fitting approach is inspired by DualSDF [28]. The main difference to their approach is that we fit a single primitive representation to a single mesh instead of learning an entire latent space of fine and coarse representation for a collection of meshes. In contrast to DualSDF, we are thus solving an optimization rather than a learning problem.

Following DualSDF, we fit primitives by minimizing the difference between the signed distance fields of the primitive-based representation and the target mesh $\mathcal{M}_{closed}$. In particular, we compute the signed distance of the primitive-based representation at each of the points sampled during pre-processing and compare this distance to the sampled value. Our primitive-based representation consists of N primitives $\{\boldsymbol{\alpha}_i | i = 1, ..., N\}$ where $\boldsymbol{\alpha}_i \in \mathbb{R}^{k_i}$ is the vector of parameters describing that primitive. The dimensionality k^i denotes the degree of freedom for primitive i. For example, a sphere primitive is defined by its 3D position $\boldsymbol{c} \in \mathbb{R}^3$ and radius $r \in \mathbb{R}$, giving four degrees of freedom. Most primitives have analytical signed distance formulations, making it easy to compute the signed distance from the primitive to a point $\boldsymbol{x}$. For example, the signed distance for the sphere can be written as

$$d^{sphere}(\boldsymbol{x}, \boldsymbol{\alpha}^{sphere}) = ||\boldsymbol{x} - \boldsymbol{c}||_2 - r. \tag{5}$$

A primitive-based representation consists of a union of individual primitives. The signed distance function of this union can be written as the minimum signed distance of the individual primitives:

$$\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_N], \tag{6}$$

$$d^p(\boldsymbol{x}, \boldsymbol{\alpha}) = \min_{i \in \{1,..,N\}} d_i(\boldsymbol{x}, \boldsymbol{\alpha}_i). \tag{7}$$

We fit the primitive-based representation by searching for the parameters $\boldsymbol{\alpha}$ that minimize the difference between the primitive signed distance $d^p(\boldsymbol{x}_k, \boldsymbol{\alpha})$ and the sampled signed distance d^s_k at the sample points $\boldsymbol{x} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_K]$:

$$\hat{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{\alpha}} \sum_{k \in \{1,...,K\}} \mathcal{L}_{sdf}(d^p(\boldsymbol{x}_k, \boldsymbol{\alpha}), d^s_k). \tag{8}$$

Following DualSDF, we use a truncated squared error loss for $\mathcal{L}_{sdf}$:

$$\mathcal{L}_{sdf}(d^p, d^s) = \begin{cases} (d^p - d^s)^2, & \text{for } d^s \geq 0 \\ \max(d^p, 0), & \text{for } d^s < 0. \end{cases} \tag{9}$$

We propose optimizing the parameters $\boldsymbol{\alpha}$ with over-parameterized neural networks rather than optimizing them directly. We found this approach to be less susceptible to local minima and faster to optimize. More precisely, we formulate this over-parameterization as follows:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \sum_{k \in \{1,...,K\}} \mathcal{L}_{sdf}(d^p(\boldsymbol{x}_k, \boldsymbol{\alpha}), d^s_k) \\ \text{s.t.} \quad & \boldsymbol{\alpha} = g_{\boldsymbol{\theta}}(), \end{aligned} \tag{10}$$

where $g_{\boldsymbol{\theta}}$ is a neural network with parameters $\boldsymbol{\theta}$ that takes no input and outputs the desired parameters $\boldsymbol{\alpha}$. We optimize over $\boldsymbol{\theta}$ using gradient descent instead of directly optimizing over $\boldsymbol{\alpha}$. This is an over-overparameterized regime as $dim(\boldsymbol{\theta}) \gg dim(\boldsymbol{\alpha})$, where $dim()$ refers to the vector's dimensionality. The architecture for $g_{\boldsymbol{\theta}}$ is adapted from DualSDF and shown in Fig. 3.

We only sample signed distance values close to the mesh. Hence, with the above formulation, it is possible that a primitive-based representation achieves zero loss even though it contains some primitives that are far from the mesh. We prevent this scenario by penalizing primitives whose centers are outside of the unit sphere, using the subsequent regularization loss:

$$\mathcal{L}_{reg_c} = \begin{cases} ||\mathbf{c}||_2^2 - 1, & \text{for } ||\mathbf{c}||_2^2 > 1 \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

It is desirable to allow rotations for some primitives, such as boxes, to achieve a more expressive representation. We chose to parameterize such rotations using unit Quaternions $\boldsymbol{q} = [qw, qx, qy, qz]$ as they can be smoothly interpolated. We enforce the unit constraint using regularization:

$$\mathcal{L}_{reg_q} = (||\boldsymbol{q}||_2 - 1)^2. \tag{12}$$

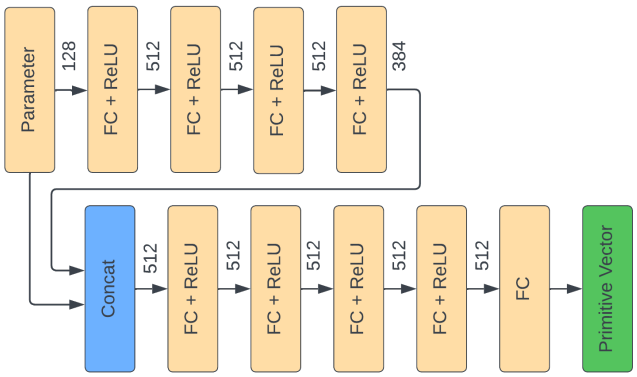

Fig. 3: The primitive fitting network architecture. The network takes no data input and outputs the primitive attribute vector. A parameter vector is passed through a fully connected network with a dense connection. *FC* denotes fully connected linear layer and *Concat* denotes a conactenation operation. The numbers denote the size of the vector after each layer, where the desired primitive output determines the size of the last/ primitive vector.

For primitives with symmetries, such as boxes, the optimization dynamics can be significantly improved by constraining the allowed rotation range to prevent these symmetries. For example, for a box primitive parameterized by its center, width, length, height, and Quaternion, we constrain the associated axis angle to the range $[-\pi/4, \pi/4]$:

$$\mathcal{L}_{reg_r} = \begin{cases} \tau - \pi/4, & \text{for } \tau > \pi/4 \\ -\tau - \pi/4, & \text{for } \tau < -\pi/4 \\ 0, & \text{otherwise,} \end{cases} \tag{13}$$

where $\tau = 2\arccos(qw)$ is the axis angle corresponding to the Quaternion. During optimization, we minimize a weighted combination of the individual loss terms:

$$\mathcal{L} = \mathcal{L}_{sdf} + w_c\mathcal{L}_{reg_c} + w_q\mathcal{L}_{reg_q} + w_r\mathcal{L}_{reg_r}. \tag{14}$$

In practice, we use $w_c = w_q = 0.001$ and $w_r = 0.1$.

D. Physical property estimation

More than good geometry is needed for accurate physics simulations as the object's physical properties $\mathcal{P}$, such as its mass, center of mass, and moment of inertia, significantly impact the dynamics. However, it is much harder to estimate these properties from images alone, as objects might look the same from the outside but differ significantly in their physical properties. We propose to manually measure the object's mass m using a scale and use this mass to obtain the object's moment of inertia with a uniform density assumption. We first compute the triangle mesh's bounding box. We then use the ray stabbing method described in section IV-C2 to densely sample signed distance values inside this bounding box. We only keep the samples with negative signed distance, giving us a dense sample representation of the object volume. The raw NeRF mesh may have some floaters, i.e., triangle clusters, around the object mesh. We perform outlier rejection using nearest-neighbor queries to prevent these floaters from skewing our inertia estimates. In particular, for every negative distance sample, we compute the number of neighbors within a radius r of that sample and reject every sample with less than M neighbors. We then take discrete estimates of the integrals that define the center of mass $\boldsymbol{\mu}$ and moment of inertia $\boldsymbol{J}$:

$$\boldsymbol{\mu} = \frac{1}{m}\int_{\boldsymbol{x}\in\mathbb{R}^3} \boldsymbol{x}\phi(\boldsymbol{x})\, d\boldsymbol{x} \approx \frac{1}{m}\sum_{\boldsymbol{x}\in X} \boldsymbol{x}\phi(\boldsymbol{x}) \tag{15}$$

$$\begin{aligned} \boldsymbol{J} &= \int_{\boldsymbol{x}\in\mathbb{R}^3} \phi(\boldsymbol{x})(\bar{\boldsymbol{x}}^T\bar{\boldsymbol{x}}I - \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^T)\, d\boldsymbol{x} \\ &\approx \sum_{\boldsymbol{x}\in X} \phi(\boldsymbol{x})(\bar{\boldsymbol{x}}^T\bar{\boldsymbol{x}}I - \bar{\boldsymbol{x}}\bar{\boldsymbol{x}}^T), \end{aligned} \tag{16}$$

where $\bar{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{\mu}$ are the sample coordinates relative to the center of mass, and X is the set of sampled 3D coordinates that remain after outlier rejection. $\phi(\boldsymbol{x})$ is the density at sample point $\boldsymbol{x}$ which we take as $\phi(\boldsymbol{x}) = \frac{m}{|X|}, \forall \boldsymbol{x}$ by our uniform density assumption.

E. URDF Construction

Our final object URDF consists of a visual geometry $\mathcal{V}$, a collision geometry $\mathcal{C}$, and physical properties $\mathcal{P}$. We take the textured NeRF mesh as the visual geometry, the primitive-based mesh approximation as the collision geometry, and the estimated mass, center of mass, and moment of inertia as the physical properties. We add the primitive-based representation as individual primitives rather than computing a triangle mesh union of the primitives ourselves.

V. Results

We proposed an automated Real2Sim system for creating dynamically accurate simulation assets for real-world objects. In particular, given a real-world object and an estimate of its mass, our pipeline produces a simulation asset for the object. This asset comprises collision geometry, visual geometry, and physical properties. We run extensive simulation experiments to evaluate our pipeline's performance in terms of visual reconstruction and simulation accuracy. We place a particular emphasis on evaluating the tradeoffs in simulation performance that can be achieved with different instances of our primitive-based collision geometries.

A. Experiment Details

1) Datasets: To demonstrate our real2sim pipeline, we use real-world data of the mustard bottle and the tomato soup can from the YCB Object Dataset [37]. We show additional results on the project page.

We used our robotics system to collect multi-view RGB images of the mustard bottle and tomato soup can, as described in section IV-A. The result was 219 images whose views were limited by the robot's workspace constraints. Fig. 4 shows the camera poses associated with these images for the mustard

bottle. The poses are the same for the tomato soup can. Due to the workspace limits, the poses are more sparse in some areas and more dense in others. Our results show that our pipeline can deal with this type of view sparsity.

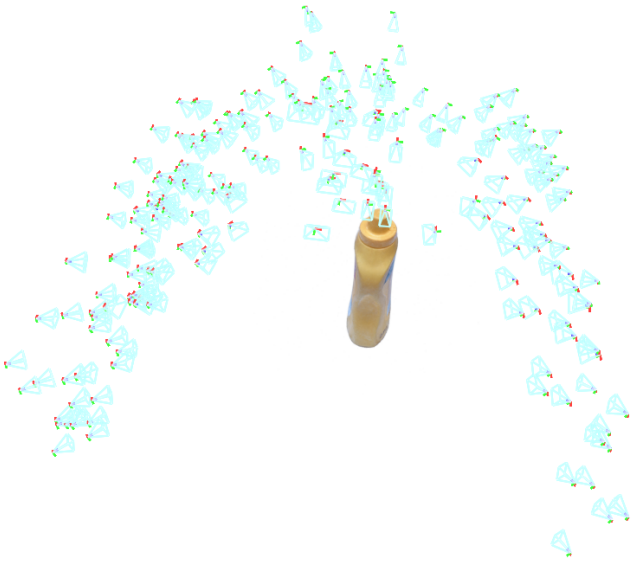

Fig. 4: The camera poses that correspond to the images that were collected by our robotics system for the YCB mustard bottle. The triangles are the camera frustums, showing the cameras' field of view.

2) Simulation Environment: We chose Drake [11] as our physics simulator for simulating our simulation assets due to its focus on accurate dynamics. Drake provides two contact models for computing contact forces during the simulation; point contact and Hydroelastic [38]. Point contact is the contact model supported by most state-of-the-art simulators, while Hydroelastic is a soft contact model specific to Drake. We evaluate our collision geometries using both of these contact models.

B. Geometric and Visual Reconstruction

Fig. 5 shows the textured triangle mesh of the YCB mustard bottle, as obtained from Nerfstudio. The high visual quality indicates our robot system's ability to record high-quality visual data. Fig. 6 shows the raw NeRF mesh of the mustard bottle and various geometric approximations of it. CoACD is a state-of-the-art convex decomposition approach [14]. The remaining geometric approximations are a representative selection of our primitive-based approximations. It can be observed that we are able to choose the desired fidelity of the approximation by choosing the number of primitives. For example, the representation consisting of 10 cylinders is a coarse approximation of the mustard bottle, while the representation consisting of 100 boxes achieves significantly higher fidelity.

C. Simulation Results

We quantitatively evaluate our simulation assets using two simulated tasks, *Planar Pushing* and *Actuated Ramp*, and

Fig. 5: The textured triangle mesh from NeRF of the YCB mustard bottle.

two objects, the YCB mustard bottle and tomato soup can. We simulate a representative sample of our primitive-based collision geometries for each task and compare them to convex decomposition as a baseline. Convex decomposition is the most popular mesh simplification algorithm for simulation. In particular, we compare against the state-of-the-art convex decomposition method CoACD [14]. For the comparison, we use our real2sim pipeline but replace our mesh simplification component with CoACD.

To quantitatively evaluate simulation performance, we simulate a manually cleaned version of the raw NeRF mesh using Drake's Hydroelastic contact model and treat this as the ground truth. This ground truth is sub-optimal as we have no guarantee that this mesh represents ground truth dynamic behavior. However, obtaining an exact ground truth would require comparing simulation roll-outs to real-world roll-outs. Doing so would require building a dynamically accurate digital replica of an entire real-world scene and ensuring identical initial conditions when comparing roll-outs. Both are open problems and beyond the scope of this work. Nevertheless, our experiments showed no human-noticeable dynamic artifacts when simulating the manually cleaned NeRF mesh with Hydroelastic. Moreover, the NeRF mesh is of much higher fidelity than the simplified geometries (CoACD and primitive-based), which might lead to dynamics closer to the desired real-world dynamics.

We evaluate the equation error performance, which is the one-step prediction error. We compute the equation error by setting the simulation asset's state (translation, orientation, and velocity) equal to the ground truth state every 0.1 seconds. For completeness, our project page additionally contains simulation error results. Simulation error is the long-term prediction error computed by rolling out the entire simulated trajectory without resets to the true state. We chose the classical average

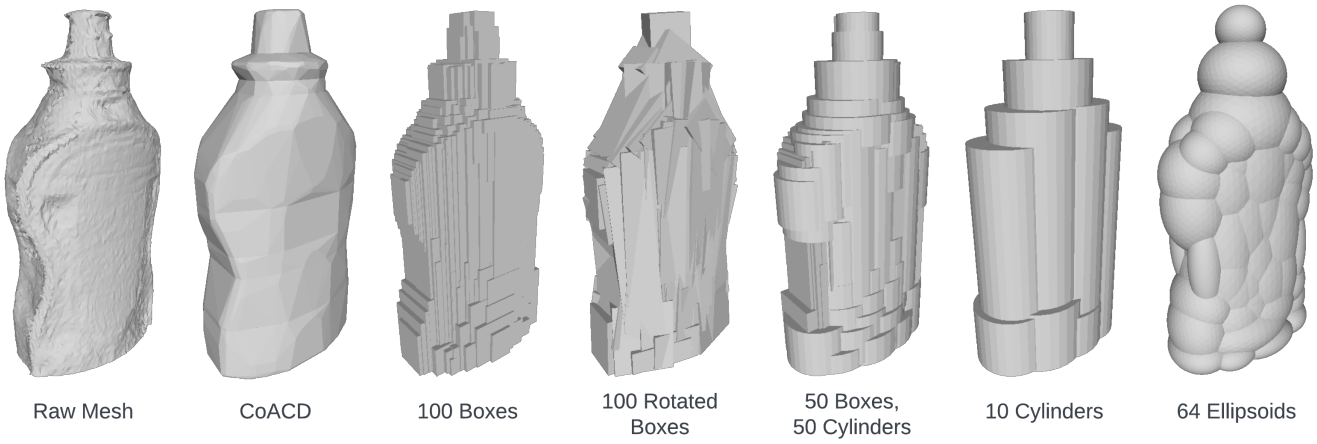

Fig. 6: The raw NeRF mesh of the YCB mustard bottle and different collision geometry approximations of it.

translation error (ATE) and an orientation-considered average error (OCAE) as metrics for evaluating dynamical accuracy. The ATE is computed as

$$ATE = \frac{1}{N} \sum_{n \in \{1,\dots,N\}} ||\boldsymbol{t}[n] - \hat{\boldsymbol{t}}[n]||_2 \tag{17}$$

where $\boldsymbol{t}[n]$ is the ground-truth translation and $\hat{\boldsymbol{t}}[n]$ the actual translation at timestep n. In practice, we use a timestep duration of 0.001 seconds. It is hard to directly include translation and orientation errors in a single metric as translation and orientation are expressed in different units. Hence, we propose to use what we call an orientation-considered average error. In particular, we use the fact that three linearly independent points fully determine an orientation and sample three such points relative to the object frame. We then compute the ATE for each of the three points and take the average as the OCAE.

Next to dynamical accuracy, contact point continuity is also important. We prefer if the contact forces acting on a geometry vary smoothly rather than discontinuously. Discontinuous contact forces make obtaining the contact force gradients difficult, which can be desirable for downstream applications. Moreover, discontinuous contact forces can make simulations harder, requiring smaller timesteps and, consequently, longer simulation times. We propose to quantify contact point continuity by first expressing the contact points relative to the object pose and then computing the average magnitude of these relative contact points. The result is a single scalar per timestep that contains information about the contact point locations relative to the object. We then take the gradient of this time series using second-order central differences. Our contact force discontinuity metric is the average of these gradients, where a higher value indicates that the contact points are more discontinuous.

For every collision geometry, we run ten independent simulations for every object and task combination, yielding a total of 40 simulations per collision geometry. We report the average metrics of these 40 simulations.

1) Planar Pushing: The *Planar Pushing* task involves a simulation asset being placed on a planar floor and pushed with a sphere, as shown in Fig. 7. The sphere starts away from the object and slowly moves toward the object's center. The task terminates after a pre-defined timeout. This task is dynamically interesting, as it involves varying contact interactions of the object with the floor and the sphere.

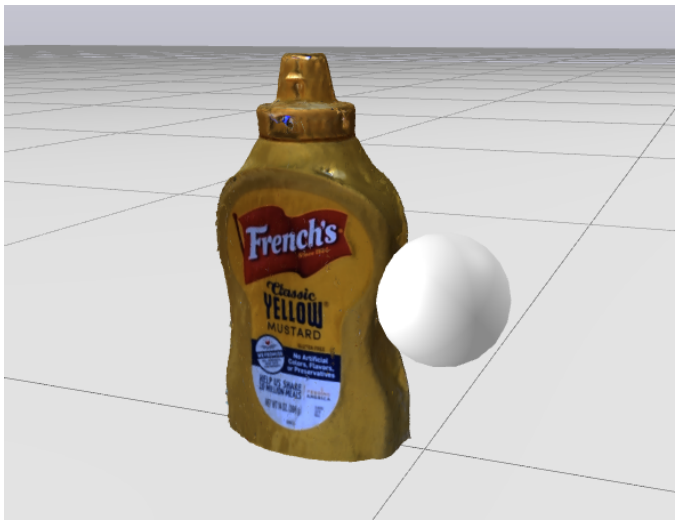

Fig. 7: The *Planar Pushing* task. A simulation asset of the YCB mustard bottle is pushed by a sphere along a planar surface.

2) Actuated Ramp: The *Actuated Ramp* task involves a simulated asset being placed on a flat ramp. The ramp then quickly moves into an inclined position, leading to the asset rolling down the ramp, as shown in Fig. 8. The task terminates after a pre-defined timeout. This task is highly dynamic as the object rolls down the ramp at high accelerations. Consequently, minor differences in dynamic behavior can lead to significant errors after a few timesteps. Subtle geometric differences

might result in very different dynamic behaviors, such as an object sliding rather than rolling down the ramp.

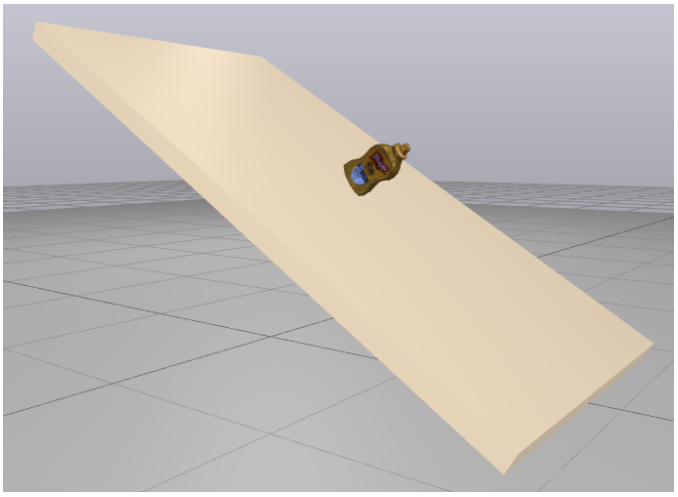

Fig. 8: The *Actuated Ramp* task. A simulation asset of the YCB mustard bottle rolls down the ramp.

3) Findings: Our experiments showed that our pipeline is capable of producing simulation assets that are highly dynamically accurate. For example, our primitive-based representation consisting of 400 boxes achieved an average ATE of $1.34mm$ and an average OCAE of $1.74cm$. This is low considering how dynamic the task is and that the simulation asset's state is only reset to the true state every 0.1 seconds. CoACD only achieved an average ATE of $2.60mm$ and an average OCAE of $3.31cm$, indicating a 48.5% and 47.4% improvement over CoACD, respectively. A complete results table of the averaged metrics can be found in the appendix, and results for the individual experiments can be found on the project page.

Moreover, we found a tradeoff between simulation accuracy and simulation time, i.e., representations with higher fidelity can achieve a higher simulation accuracy but take longer to simulate than simpler representations. Fig. 9 shows a Pareto frontier plot of the tradeoff between OCAE and simulation time. Representations at the top-right boundary represent different optimal tradeoffs. Representations that lie between the bottom-left corner and that boundary are sub-optimal. For example, both the *boxes_150_hydroelastic* and the *cylinders_10_hydroelastic* representations are optimal tradeoffs; *boxes_150_hydroelastic* achieves a lower OCAE but *cylinders_10_hydroelastic* achieves a lower simulation time. However, *coacd_hydroelastic* is sub-optimal as it is strictly worse than *cylinders_10_hydroelastic* in both OCAE and simulation time. Our primitive-based representations enable choosing between many simulation accuracy/ simulation time tradeoffs. This is useful as different downstream applications might have different preferences for balancing dynamic accuracy and simulation time. The Pareto frontier plot additionally reveals some general trends that give insight into what makes good collision geometries for dynamically accurate and fast simulations. The same representations tend to achieve higher simulation accuracy (lower OCAE) when simulated using Drake's Hydroelastic contact model than with a point contact model. However, the point contact model tends to lead to faster simulations. For Hydroelastic contact, coarse primitives such as boxes and cylinders tend to outperform finer geometries such as spheres and ellipsoids. For point contact, the opposite tends to be the case, where finer geometries outperform coarser ones. A likely reason for this is that spheres and ellipsoids score highly in terms of contact point continuity which is more important for hard contact models, such as point contact, than for soft contact models, such as Hydroelastic.

Regarding contact point continuity, we found that our primitive-based representations outperform CoACD by significant margins. The representation consisting of 128 spheres achieved a CPD score of 0.0025 while CoACD only achieved a score of 0.0093. This represents a 73.1% improvement over CoACD. In general, sphere-based representations tend to perform particularly well in terms of contact force continuity. Similarly to the dynamic accuracy/ simulation time tradeoff, there is a tradeoff between contact force continuity and dynamic accuracy/ simulation time, creating a three-way tradeoff.

4) Recommendations: When choosing which collision geometry to use with our Real2Sim pipeline, we recommend making a choice based on the desired simulation accuracy/ time tradeoff. If contact point continuity is vital, we recommend starting with the desired simulation accuracy/ time tradeoff and adjusting it slightly based on contact point continuity differences. We provide a variety of Pareto frontier plots and a precise results table in the appendix to facilitate such decision-making. Generally, representations consisting of 20 to 150 boxes or cylinders perform well when simulated using Drake's Hydroelastic contact model. Some shapes are more easily approximated with primitive shapes than others. Consequently, for some shapes, fewer primitives might work well, while in the general case, more primitives are necessary for achieving high simulation accuracy.

VI. Limitations and Future Work

We demonstrated that our pipeline produces dynamically accurate simulation assets when treating the raw NeRF mesh as the ground truth mesh. However, it needs to be clarified how accurate this assumption is. Specifically, the most accurate geometric mesh might not be the best dynamic mesh. We are excited about running real-world experiments to compare the simulation behavior to the actual real-world behavior.

Our system computes the object's moment of inertia and center of mass using a constant density assumption. This method only works well for objects that have close to constant densities. In the future, we plan to use the *Panda Emika Franka* robot to identify the object's physical properties through robot interaction.

We use a robot system to collect multi-view RGB images of the object of interest. Currently, this system does not interact with the object and hence cannot observe the occluded object bottom. Our solution of assuming the object's bottom to be

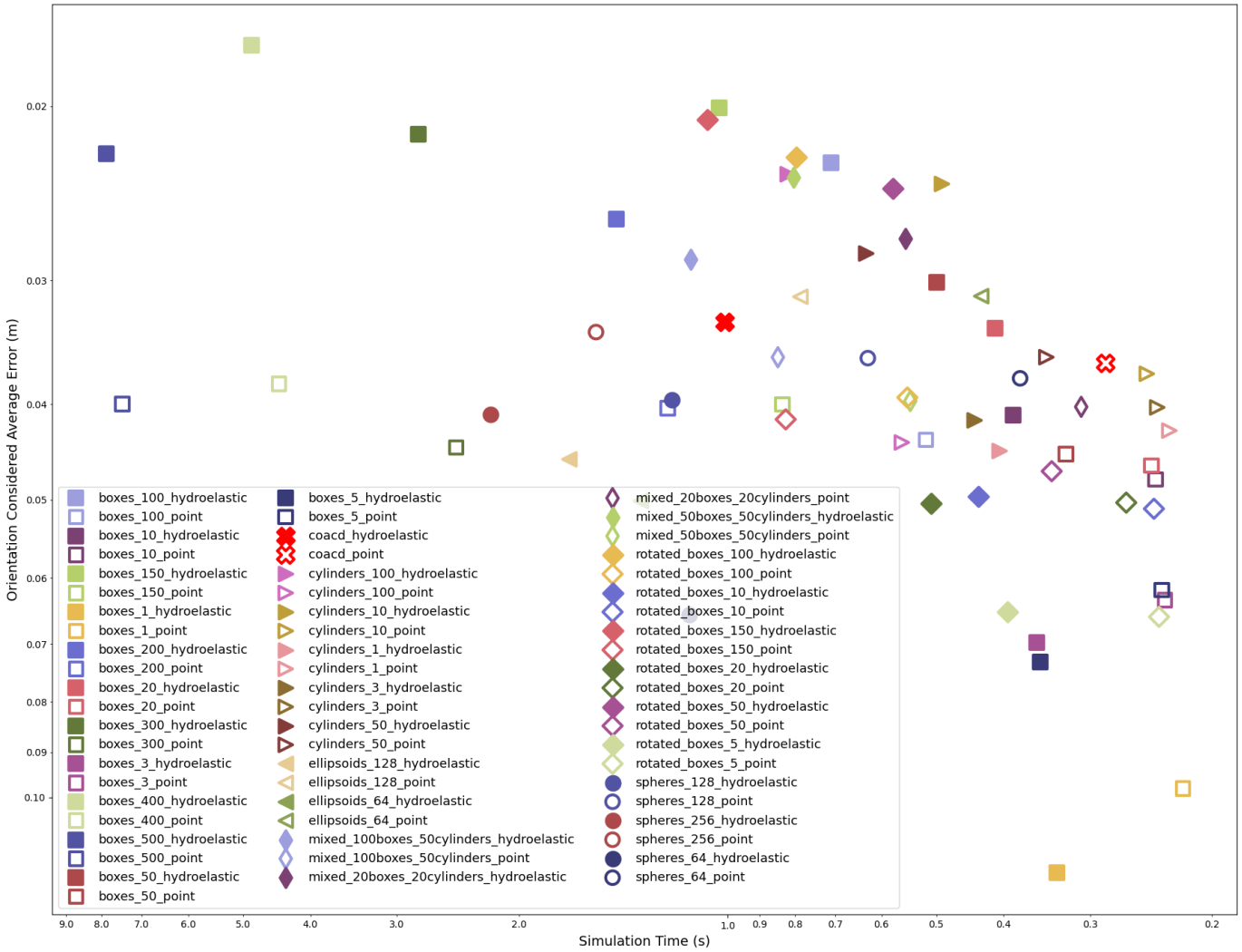

Fig. 9: A Pareto frontier plot showing the average tradeoff between orientation-considered average error (OCAE) and simulation time for the equation error experiments with the YCB mustard bottle and tomato soup can. Representations get better in simulation time from left to right and better in OCAE from bottom to top. Both axes use a log scale.

flat might only work for some objects. Consequently, a fruitful direction of future work would be to change the system to one where the camera is static, and the robot moves the object in front of the camera until every part of the object has been observed. The reconstruction could then be performed using a method that jointly tracks the object's pose and reconstructs its geometry, such as [18].

VII. Conclusion

In this work, we presented an automated Real2Sim pipeline for creating dynamically accurate simulation assets for real-world objects. We additionally proposed a novel method for finding a collision geometry composed of primitive shapes. Our simulation results demonstrate that our system is capable of producing simulation assets that are both dynamically accurate and fast to simulate. Furthermore, we show that there exists an entire collection of primitive-based representations that yield different tradeoffs in terms of simulation accuracy, simulation time, and contact point continuity. Our primitive-based representations can outperform state-of-the-art geometric simplification approaches in all of these metrics.

Acknowledgment

We thank Ria Sonecha, Andy Lambert, Terry Suh, Ge Yang, and Phillip Isola for insightful discussions and feedback.

References

[1] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, "Legged locomotion in challenging terrains using egocentric vision," in *6th Annual Conference on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=Re3NjSwf0WF

[2] W. Zhou and D. Held, "Learning to grasp the ungraspable with emergent extrinsic dexterity," in *Conference on Robot Learning (CoRL)*, 2022.

[3] Y. Qin, B. Huang, Z.-H. Yin, H. Su, and X. Wang, "Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation," *Conference on Robot Learning (CoRL)*, 2022.

[4] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," 2017.
[5] D. Ho, K. Rao, Z. Xu, E. Jang, M. Khansari, and Y. Bai, "Retinagan: An object-aware approach to sim-to-real transfer," 2021.
[6] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, "Auto-tuned sim-to-real transfer," 2021. [Online]. Available: https://arxiv.org/abs/2104.07662
[7] V. Lim, H. Huang, L. Y. Chen, J. Wang, J. Ichnowski, D. Seita, M. Laskey, and K. Goldberg, "Planar robot casting with real2sim2real self-supervised learning," 2021. [Online]. Available: https://arxiv.org/abs/2111.04814
[8] L. Wang, R. Guo, Q. Vuong, Y. Qin, H. Su, and H. Christensen, "A real2sim2real method for robust object grasping with neural surface reconstruction," 2022. [Online]. Available: https://arxiv.org/abs/2210.02685
[9] A. Byravan, J. Humplik, L. Hasenclever, A. Brussee, F. Nori, T. Haarnoja, B. Moran, S. Bohez, F. Sadeghi, B. Vujatovic, and N. Heess, "Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields," 2022. [Online]. Available: https://arxiv.org/abs/2210.04932
[10] S. L. Cleac'h, H.-X. Yu, M. Guo, T. A. Howell, R. Gao, J. Wu, Z. Manchester, and M. Schwager, "Differentiable physics simulation of dynamics-augmented neural objects," 2022.
[11] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu
[12] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.
[13] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.
[14] X. Wei, M. Liu, Z. Ling, and H. Su, "Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search," *arXiv preprint arXiv:2205.02961*, 2022.
[15] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *NeurIPS*, 2021.
[16] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.
[17] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
[18] B. Wen, J. Tremblay, V. Blukis, S. Tyree, T. Muller, A. Evans, D. Fox, J. Kautz, and S. Birchfield, "Bundlesdf: Neural 6-dof tracking and 3d reconstruction of unknown objects," *CVPR*, 2023.
[19] M. Worchel, R. Diaz, W. Hu, O. Schreer, I. Feldmann, and P. Eisert, "Multi-view mesh reconstruction with neural deferred shading," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.
[20] R. Chen, S. Han, J. Xu, and H. Su, "Visibility-aware point-based multi-view stereo network," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10, pp. 3695–3708, 2021.
[21] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2mesh: Generating 3d mesh models from single rgb images," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 55–71.
[22] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, J. Kerr, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, and A. Kanazawa, "Nerfstudio: A modular framework for neural radiance field development," *arXiv preprint arXiv:2302.04264*, 2023.
[23] Z. Tang, B. Sundaralingam, J. Tremblay, B. Wen, Y. Yuan, S. Tyree, C. Loop, A. Schwing, and S. Birchfield, "RGB-only reconstruction of tabletop scenes for collision-free manipulator control," in *ICRA*, 2023.
[24] W. Schroeder, J. Zarge, and W. Lorensen, "Decimation of triangle meshes," *SIGGRAPH Comput. Graph.*, vol. 26, pp. 65–70, 06 1997.
[25] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. USA: ACM Press/Addison-Wesley Publishing Co., 1997, p. 209–216. [Online]. Available: https://doi.org/10.1145/258734.258849
[26] E. L. Khaled Mamou and A. Peters, "Volumetric hierarchical approximate convex decomposition," in *Game Engine Gems 3*. AK Peters, 2016, p. 141–158.
[27] F. Sun, Y.-K. Choi, Y. Yu, and W. Wang, "Medial meshes for volume approximation," 2013.
[28] Z. Hao, H. Averbuch-Elor, N. Snavely, and S. Belongie, "Dualsdf: Semantic shape manipulation using a two-level representation," pp. 7631–7641, 2020.
[29] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik, "Learning shape abstractions by assembling volumetric primitives," in *Computer Vision and Pattern Regognition (CVPR)*, 2017.
[30] I. A. Sucan and S. Chitta, "Moveit," 2019. [Online]. Available: https://moveit.ros.org
[31] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
[32] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017, cite arxiv:1703.06870Comment: open source; appendix on more results. [Online]. Available: http://arxiv.org/abs/1703.06870
[33] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 163–169. [Online]. Available: https://doi.org/10.1145/37401.37422
[34] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.
[35] J. Young, "xatlas," 2022. [Online]. Available: https://github.com/jpcy/xatlas
[36] F. Nooruddin and G. Turk, "Simplification and repair of polygonal models using volumetric techniques," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 191–205, 2003.
[37] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 510–517.
[38] J. Masterjohn, D. Guoy, J. Shepherd, and A. Castro, "Velocity level approximation of pressure field contact patches," 2021. [Online]. Available: https://arxiv.org/abs/2110.04157
[39] H. Nguyen, *Gpu Gems 3*, 1st ed. Addison-Wesley Professional, 2007.

APPENDIX

This appendix presents additional geometric and simulation results for the YCB mustard bottle and the YCB tomato soup can.

Comparing sphere fitting to particle representations. Fitting spheres is similar to the classical particle representations used in GPU-powered physics simulators as proposed in GPU Gems 3 [39]. Such particle representations get constructed by discretizing the space around the mesh and then inserting a sphere in each voxel if it is inside the mesh (negative signed distance). Fig. 10 compares optimization-based primitive fitting with such particle approximations at different discretization fidelities. The figure demonstrates how fitting spheres via optimization is more efficient than allocating particles based on a yes/ no decision. More spheres/ particles are required to achieve an equivalent fidelity as the optimization-based approach can. In particular, optimization achieved equivalent fidelity with 256 spheres than the yes/ no particle representation achieved with ten thousand spheres. Optimization can use varying radii and move spheres around through a continuous space. This enables more efficient usage

of a limited number of spheres, which is essential for achieving high simulation speeds.

Additional Pareto frontier plots. Fig. 11 shows the Pareto frontier plot for the tradeoff between average translation error and simulation time. Fig. 12 shows the Pareto frontier plot for the tradeoff between contact point discontinuity and orientation-considered average error. Fig. 13 shows the Pareto frontier plot for the tradeoff between the contact point discontinuity and simulation time. These plots contain average metrics from the YCB mustard bottle and tomato soup can for the *Planar Pushing* and *Actuated Ramp* experiments.

Simulation metric tables. Table I shows the average metrics from the YCB mustard bottle and tomato soup can for the *Planar Pushing* and *Actuated Ramp* experiments. OCAE stands for orientation-considered average error, ATE stands for average translation error, CPD stands for contact point discontinuity, and ST stands for simulation time. OCAE and ATE are expressed in meters, CPD in meters per second, and ST in seconds. All metrics represent the average metric of 40 separate simulations as explained in section V-C. The top score for each metric is highlighted in bold.

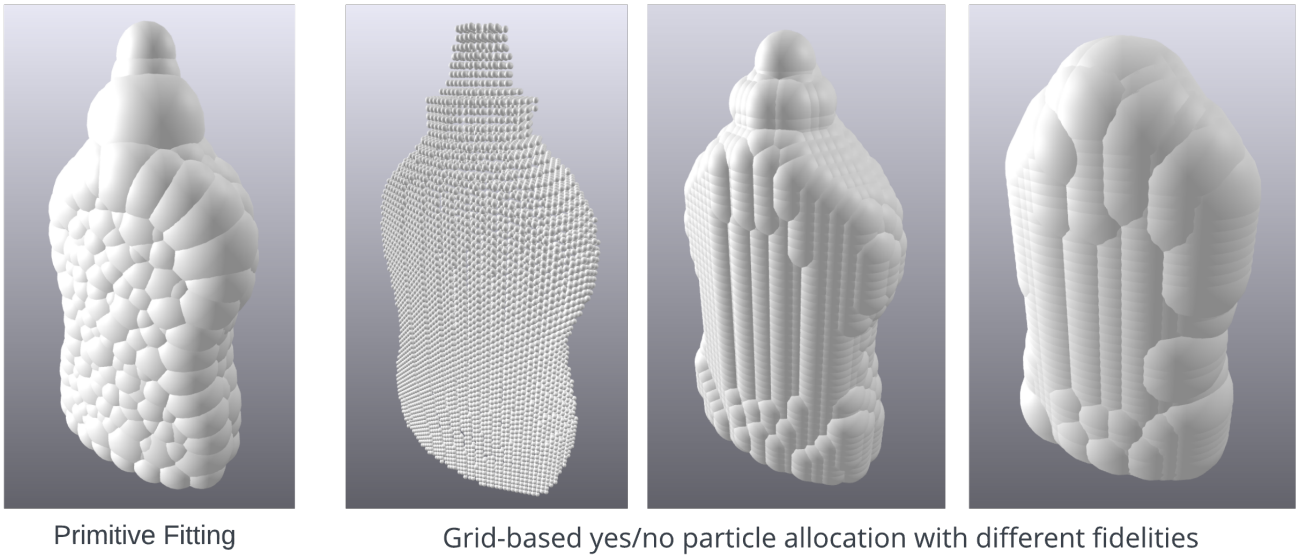

Fig. 10: Our sphere-based representation compared to Nvidia GPU Gems 3's binary particle allocation at different discretization fidelities.

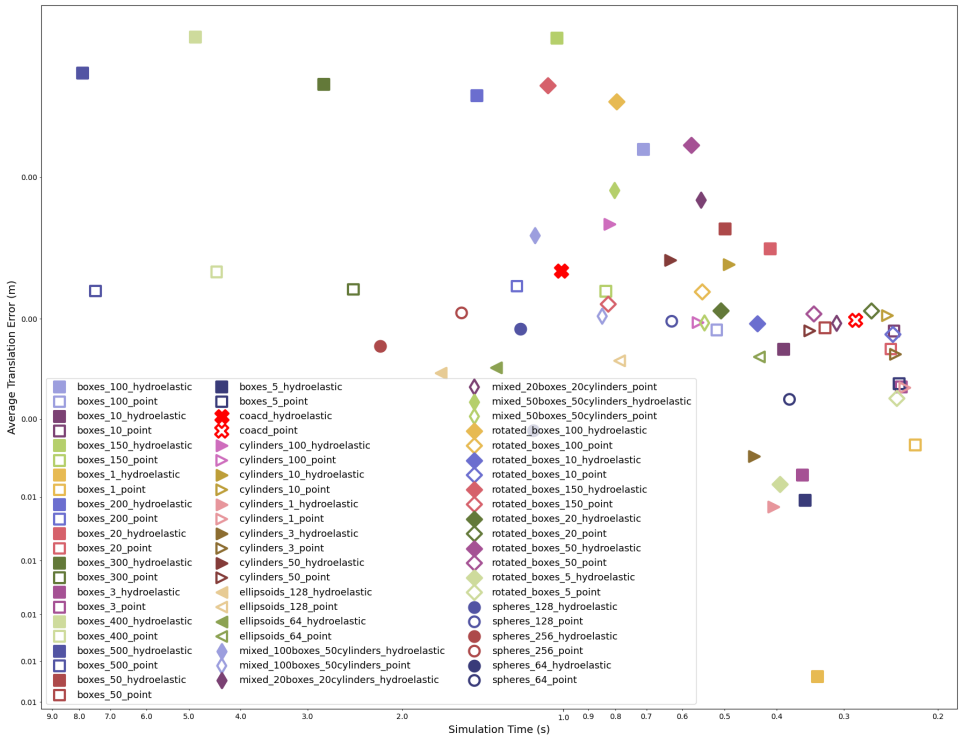

Fig. 11: A Pareto frontier plot showing the tradeoff between the average translation error (ATE) and simulation time for the equation error experiments with the YCB mustard bottle and tomato soup can.

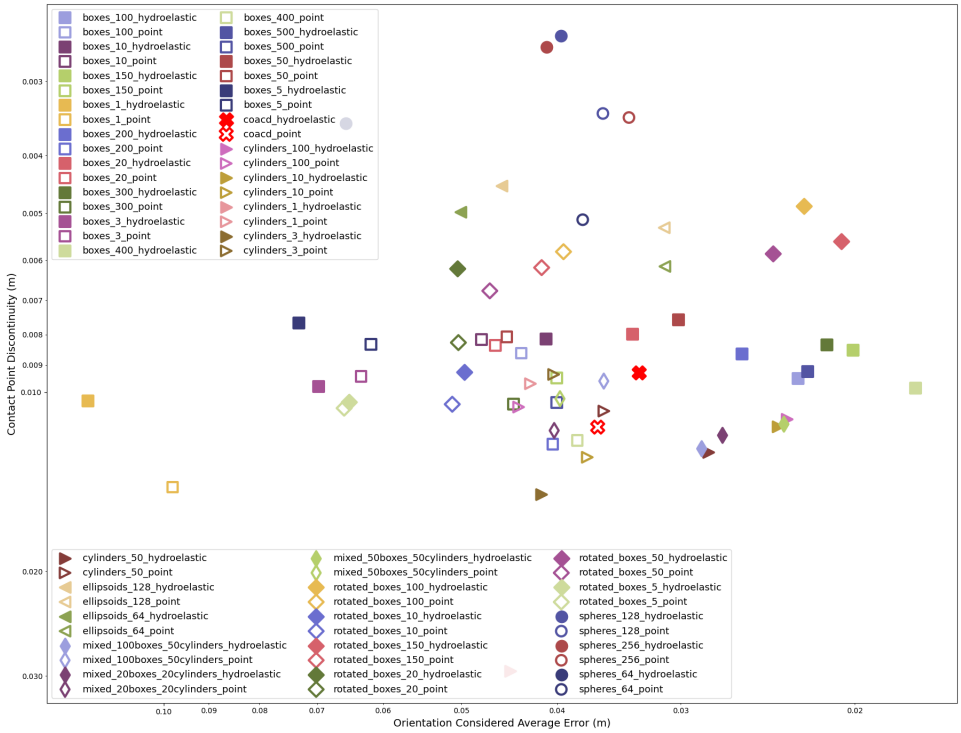

Fig. 12: A Pareto frontier plot showing the tradeoff between the contact point discontinuity (CPD) and orientation-considered average error (OCAE) for the equation error experiments with the YCB mustard bottle and tomato soup can.

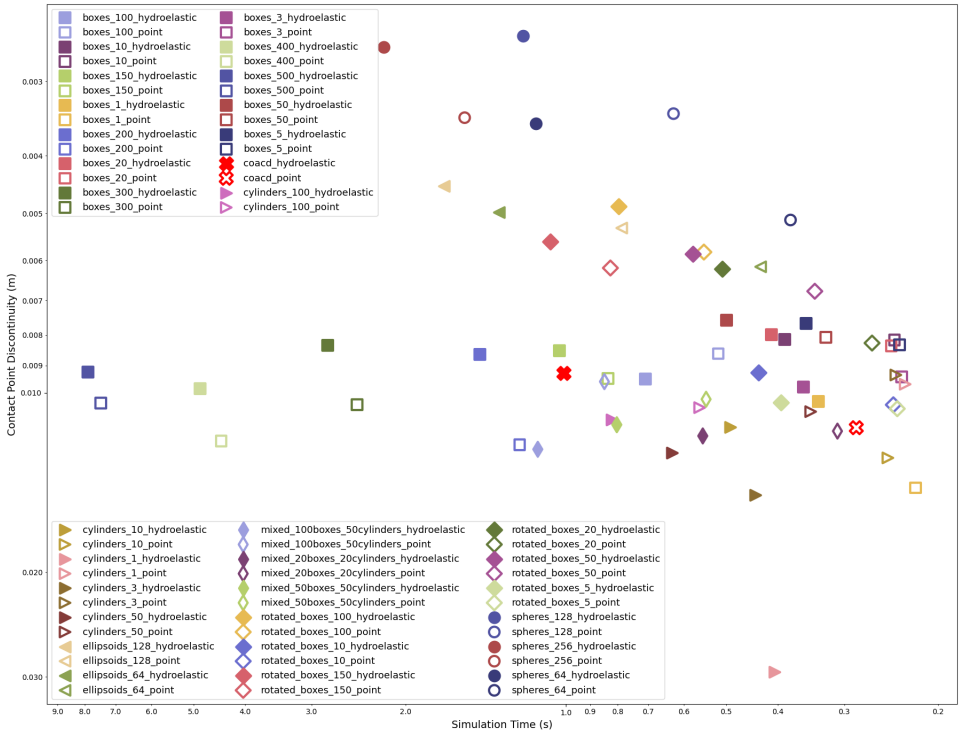

Fig. 13: A Pareto frontier plot showing the tradeoff between the contact point discontinuity (CPD) and simulation time for the equation error experiments with the YCB mustard bottle and tomato soup can.

Representation	OCAE	ATE	CPD	ST
boxes_100_hydroelastic	0.0228	0.0018	0.0095	0.7094
boxes_100_point	0.0435	0.0031	0.0086	0.5175
boxes_10_hydroelastic	0.0410	0.0033	0.0081	0.3877
boxes_10_point	0.0477	0.0031	0.0082	0.2415
boxes_150_hydroelastic	0.0201	0.0013	0.0085	1.0291
boxes_150_point	0.0400	0.0028	0.0095	0.8332
boxes_1_hydroelastic	0.1192	0.0084	0.0103	0.3353
boxes_1_point	0.0980	0.0043	0.0144	**0.2203**
boxes_200_hydroelastic	0.0260	0.0016	0.0086	1.4485
boxes_200_point	0.0404	0.0027	0.0122	1.2205
boxes_20_hydroelastic	0.0335	0.0025	0.0080	0.4114
boxes_20_point	0.0462	0.0033	0.0083	0.2449
boxes_300_hydroelastic	0.0214	0.0015	0.0083	2.7964
boxes_300_point	0.0443	0.0028	0.0105	2.4647
boxes_3_hydroelastic	0.0697	0.0047	0.0098	0.3580
boxes_3_point	0.0631	0.0036	0.0094	0.2340
boxes_400_hydroelastic	**0.0174**	**0.0013**	0.0099	4.8621
boxes_400_point	0.0382	0.0026	0.0121	4.4377
boxes_500_hydroelastic	0.0223	0.0015	0.0092	7.8866
boxes_500_point	0.0400	0.0028	0.0104	7.4662
boxes_50_hydroelastic	0.0302	0.0023	0.0076	0.4992
boxes_50_point	0.0450	0.0031	0.0081	0.3249
boxes_5_hydroelastic	0.0730	0.0051	0.0076	0.3539
boxes_5_point	0.0617	0.0036	0.0083	0.2364
coacd_hydroelastic	0.0331	0.0026	0.0093	1.0091
coacd_point	0.0364	0.0030	0.0114	0.2848
cylinders_100_hydroelastic	0.0234	0.0023	0.0111	0.8201
cylinders_100_point	0.0438	0.0030	0.0106	0.5607
cylinders_10_hydroelastic	0.0239	0.0026	0.0114	0.4915
cylinders_10_point	0.0373	0.0030	0.0129	0.2485
cylinders_1_hydroelastic	0.0446	0.0051	0.0294	0.4058
cylinders_1_point	0.0426	0.0037	0.0097	0.2306
cylinders_3_hydroelastic	0.0415	0.0045	0.0149	0.4411
cylinders_3_point	0.0403	0.0033	0.0093	0.2401
cylinders_50_hydroelastic	0.0281	0.0025	0.0126	0.6315
cylinders_50_point	0.0359	0.0031	0.0108	0.3469
ellipsoids_128_hydroelastic	0.0455	0.0035	0.0045	1.6927
ellipsoids_128_point	0.0312	0.0034	0.0053	0.7852
ellipsoids_64_hydroelastic	0.0501	0.0035	0.0050	1.3341
ellipsoids_64_point	0.0311	0.0033	0.0061	0.4307
mixed_100boxes_50cylinders_hydroelastic	0.0286	0.0024	0.0124	1.1303
mixed_100boxes_50cylinders_point	0.0359	0.0030	0.0096	0.8463
mixed_20boxes_20cylinders_hydroelastic	0.0272	0.0021	0.0118	0.5540
mixed_20boxes_20cylinders_point	0.0403	0.0030	0.0116	0.3090
mixed_50boxes_50cylinders_hydroelastic	0.0236	0.0021	0.0113	0.8028
mixed_50boxes_50cylinders_point	0.0397	0.0030	0.0103	0.5452
rotated_boxes_100_hydroelastic	0.0225	0.0016	0.0049	0.7960
rotated_boxes_100_point	0.0394	0.0028	0.0058	0.5505
rotated_boxes_10_hydroelastic	0.0496	0.0030	0.0093	0.4349
rotated_boxes_10_point	0.0511	0.0031	0.0105	0.2427
rotated_boxes_150_hydroelastic	0.0206	0.0015	0.0056	1.0687
rotated_boxes_150_point	0.0415	0.0029	0.0062	0.8247
rotated_boxes_20_hydroelastic	0.0504	0.0029	0.0062	0.5083
rotated_boxes_20_point	0.0503	0.0029	0.0083	0.2661
rotated_boxes_50_hydroelastic	0.0242	0.0018	0.0058	0.5779
rotated_boxes_50_point	0.0468	0.0030	0.0068	0.3408
rotated_boxes_5_hydroelastic	0.0649	0.0048	0.0104	0.3942
rotated_boxes_5_point	0.0657	0.0038	0.0106	0.2386
spheres_128_hydroelastic	0.0396	0.0031	**0.0025**	1.2028
spheres_128_point	0.0360	0.0030	0.0034	0.6278
spheres_256_hydroelastic	0.0410	0.0032	0.0026	2.1980
spheres_256_point	0.0338	0.0030	0.0035	1.5486
spheres_64_hydroelastic	0.0654	0.0041	0.0035	1.1367
spheres_64_point	0.0377	0.0038	0.0051	0.3785

TABLE I: Average simulation metrics for the equation error experiments with the YCB mustard bottle and tomato soup can.