One-Shot Real-to-Sim via End-to-End Differentiable Simulation and Rendering

Yifan Zhu[1], Tianyi Xiang[1], Aaron M. Dollar[1], and Zherong Pan[2]

***Abstract*—Identifying predictive world models for robots from sparse online observations is essential for robot task planning and execution in novel environments. However, existing methods that leverage differentiable programming to identify world models are incapable of jointly optimizing the geometry, appearance, and physical properties of the scene. In this work, we introduce a novel rigid object representation that allows the joint identification of these properties. Our method employs a novel differentiable point-based geometry representation coupled with a grid-based appearance field, which allows differentiable object collision detection and rendering. Combined with a differentiable physical simulator, we achieve end-to-end optimization of world models or rigid objects, given the sparse visual and tactile observations of a physical motion sequence. Through a series of world model identification tasks in simulated and real environments, we show that our method can learn both simulation- and rendering-ready rigid world models from only one robot action sequence. The code and additional videos are available at our project website: https://tianyi20.github.io/rigid-world-model.github.io/.**

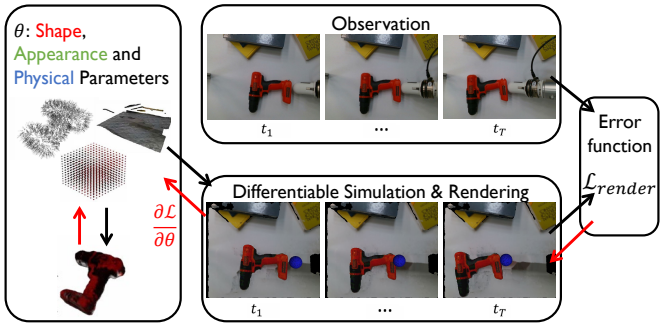


Fig. 1. From the visual and tactile observations of a single robot push (top), our method jointly optimizes the shape, appearance, and physical parameters of a world model consisting of rigid objects in the form of a rigid body simulator (bottom, the robot arm is not rendered in this picture and the end-effector is treated as a floating blue sphere robot).

I. Introduction

An accurate internal model of a robot about how its actions can affect the surrounding environment is essential for robot planning and control. Such a model, which we refer to as a *world model*, needs to render realistic raw observations such as RGB images from arbitrary viewpoints and predict consistent and accurate physical interactions. However, constructing such a model from raw observations in novel real-world environments remains challenging as it requires the identification of the geometry parameters that describe the shape of all objects (e.g. vertices and faces of a mesh), appearance parameters that define how the objects look when rendered (e.g. color and reflectance), and physical parameters (e.g. mass) of the objects in the scene. These parameters are usually partially observable, and robots are typically limited in time and computational resources.

Recently, there has been growing interest in learning world models from large offline datasets of action-labeled videos using generative modeling techniques [1], [2], [3], [4]. However, these black-box models are susceptible to distribution shifts and cannot infer properties such as the coefficient of friction. In addition, they are not physically consistent and cannot provide physical information such as contact forces, which are essential for downstream tasks. Meanwhile, an alternative approach that identifies the geometry, appearance, and physical parameters (GAP) of the environment with strong priors coming from knowledge of physics can result in a generalizable and physically consistent world model.

Many existing works employ differentiable simulators as strong physics priors that allow efficient identification of physical parameters such as inertia and coefficient of friction [5], [6], [7]. Differentiable simulators allow the gradient-based optimization of mass-inertial properties and frictional coefficients to match a physical motion sequence to sparse robot observations. However, these works assume known geometries and appearances of the objects in the scene and do not allow the algorithm to adapt the GAP simultaneously.

On the other hand, we have witnessed recent advances in learnable geometry and appearance models, such as Neural Radiance Fields (NeRF) [8] and Gaussian Splatting (GS) [9]. These methods build the rendering equation into a learnable representation to enable the identification of geometries and appearances from raw observations. However, rigid body simulators [10] typically require the use of volumetric representations with a clear definition of object surfaces such as convex hulls to detect collisions and penetration depths. Unfortunately, NeRF and GS are incompatible with the requirements of rigid body simulators since NeRF represents objects with a continuous neural field and GS with individual 3D Gaussians. To the best of the authors' knowledge, no existing method allows the simultaneous identification of the GAP properties of a world model of rigid objects from sparse robot observations.

To address these challenges, this work presents a rigid object representation that is compatible with general-purpose rigid body simulators and allows the joint optimization of GAP. As shown in Fig. 1, based on this representation, our work enables the identification of a rigid world model in the form of a full-fledged rigid body simulator from the observations of one robot push. Our proposed representation is the combination of

Manuscript received: December 8, 2024; Revised: March 7, 2025; Accepted: April 7, 2025.

This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments.

[1]Y. Zhu, T. Xiang, and A. Dollar are with the Department of Mechanical Engineering and Materials Science, Yale University, New Haven, United States. {yifan.zhu, tianyi.xiang, aaron.dollar}@yale.edu [2]Z. Pan is an independent researcher. zherong.pan.usa@gmail.com

Digital Object Identifier (DOI): see top of this page.

a recently proposed point-based shape representation Shape-as-Points (SaP) [11] and a grid-based appearance field. SaP parameterizes an object's geometry and topology using a set of surface points along with normal directions. It then uses differentiable Poisson reconstruction to recover a smooth indicator field of object occupancy, which can be converted to a mesh using a differentiable marching cubes algorithm [12]. The texture of the vertices of the mesh is then obtained by interpolating the appearance grid. Employing the mesh in a differentiable rigid body simulator [13] that provides gradients for the physical parameters and contact points of the objects, our method constructs a fully differentiable pipeline for jointly optimizing the GAP. Our contributions are:

- A jointly differentiable representation of the shape, appearance, and physical properties of rigid objects.
- An algorithm for identifying world models online from sparse robot observations, which we refer to as real-to-sim, with an end-to-end differentiable simulation and rendering pipeline.

We evaluate our method on identification problems in both simulated and real-world environments. The results show that our method can infer accurate world models from a single episode of robot interactions with the environment.

II. Related Work

Our work is closely related to differentiable rigid body simulators, learnable geometry and appearance models, and identifying world models, and we review these areas of study in this section.

A. Differentiable Rigid Body Simulator

Rigid body simulators are essential tools in robotics and engineering for testing, verification, perception, control, and planning. Traditional rigid body simulators are not differentiable, but there have been many recently proposed differentiable rigid body simulators [13], [14], [15], [16], [17], [18] for facilitating downstream system identification, robot planning, and policy optimization tasks. Different strategies are adopted to enable the calculation of gradients for the underlying non-differentiable contact dynamics, including employing a smooth contact model [18], [15], [16], using sub-gradients of the linear complementarity problem [14], and implicit gradients of nonlinear optimization [13], [17]. However, most of these methods do not provide gradients with respect to the geometry, with the exception of [17], [13], [15]. In this work, we adopt the simulator proposed by Strecke et al. [13] for its physical realism, numerical stability, and fast computation from GPU acceleration.

B. Learnable Geometry and Appearance Models

Learning 3D geometry and appearance models from 2D raw images is vital to robots' understanding of the physical world. Earlier research has focused on learning only the 3D geometries without appearance, including point-cloud-based [19] models, convex-hull-based [20] models, and learning implicit signed distance functions [21]. Neural radiance fields (NeRF) is the first method that enables a continuously learnable model for the full 3D appearance of objects and scenes, which uses neural networks to parameterize the spatial appearance properties and implicitly learn the 3D geometry. More recently, Kerbl et al. proposed Gaussian Splatting (GS) [9], a non-parametric method that represents the appearance of the scene with 3D Gaussians and significantly improves the training and rendering speeds due to their fit for fast GPU/CUDA-based rasterization. These algorithms are capable of learning detailed 3D object and scene appearances from sparse image-based observations. However, NeRF and GS lack a clear definition of rigid object surfaces as NeRF represents objects with a neural field and GS with individual 3D Gaussians. Therefore, while there are some initial attempts at integrating them with rigid body simulators [22], [23], research for robust, physically correct, and differentiable collision detections with these models is still ongoing. In addition, NeRF and GS require many diverse views of an object, which is unrealistic in typical robotic manipulation applications.

Compared to standard 3D representations such as point clouds, which do not allow volumetric collision detection, or meshes, which do not allow large geometric and topological changes during optimization, our SaP-based methods enjoy the best of both worlds. Combined with a differentiable renderer, our object representation then achieves end-to-end image-based shape optimization.

C. World Models

Traditional system identification methods [24] identify only the dynamics parameters from full state information. However, to support diverse downstream robot tasks in the real world, world models need to be built from raw observations and support both accurate dynamics prediction and photorealistic novel view synthesis. While existing works have identified world models from raw image observations using differentiable simulators [17], [13], [15], none supports simultaneous optimization of GAP. Another line of work closely related to ours is image-based generative world modeling. These works aim to predict the next RGB frame based on the current frame and action. These models are learned by training on diverse datasets with generative modeling techniques such as variational autoencoders [1], [2] and diffusion [3], [4]. The key differences between our method and these works are that our simulation, grounded in physics, is always physically consistent and is a general-purpose rigid simulator that can provide physical information such as contact forces. Purely data-driven world models generalize poorly to novel scenarios and their lack of physical information severely limits their application to downstream robot tasks. Finally, a recent work [25] proposed a method to use Gaussian Splatting along with a particle-based simulator to track and reconstruct a moving scene. Instead of identifying the physical parameters of the scene, the method optimizes virtual forces attached to each particle such that they match the observed object trajectory. Therefore, although the method can be used as a world model for prediction, the accuracy is severely limited. We include this method as a baseline in our experiments in Sec. V and demonstrate the limitation of this method.

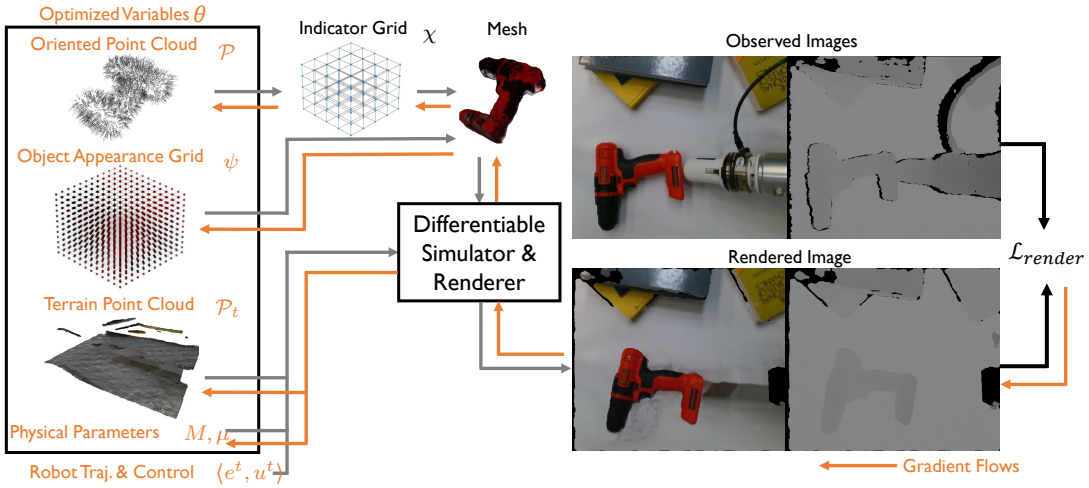


Fig. 2. Overview of the proposed fully differentiable pipeline for world model identification from sparse robot observations. Our object representation couples an oriented point cloud $\mathcal{P}$ and a 3D appearance grid ψ. Through a differentiable Poisson solver and differentiable marching cubes, the oriented point cloud is converted to an indicator grid χ and then a mesh, whose vertex textures are interpolated from the appearance grid ψ. Feeding the object mesh, physical parameters M and μ, the terrain point cloud $\mathcal{P}_t$, and the robot pushing trajectory and control $\langle e^t, u^t \rangle$ into a differentiable rigid body simulator and renderer, the predicted scenes can be rendered. Calculating the loss against observed RGB-D images, the scene shape, appearance, and physical parameters are jointly optimized with gradient descent.

III. Problem Definition

In this section, we describe our formulation of world model identification. We assume the environment consists of a rigid object and rigid terrain, whose physical properties and appearances are parameterized by θ. A robot, equipped with joint encoders and end-effector force sensors, interacts with the object at T time instances: $t_1, \cdots, t_T$, with a fixed time step δt. At each time step, the robot observes its end-effector pose $e^t \in \mathrm{SE}(3)$ and contact force $f^t \in \mathbb{R}^3$. Further, the robot is equipped with an RGB-D camera with known intrinsics that observes the object through image $o^t \in \mathbb{R}^{H \times W \times 4}$ at camera pose $c^t \in \mathrm{SE}(3)$. We further assume an image segmentation mask $m^t \in \mathbb{R}^{H \times W \times 4}$ is provided for the object, robot, and terrain. Therefore, the robot observations are a sequence $\mathcal{O} = \{\langle t, e^t, f^t, o^t, c^t, m^t \rangle\}$, and our goal is to estimate θ from the set of sparse observations $\mathcal{O}$. We formulate this problem as a physics-constrained optimization by introducing a full-fledged physics simulator function $q^{i+1}, \dot{q}^{i+1} = g(q^i, \dot{q}^i, u^i, \theta)$ that can differentiate through objects' appearance, geometry, and physical parameters. Here, q^t and $\dot{q}^{i+1}$ are the object and robot end-effector poses and velocities at timestep i and u^i is the applied robot force at the end-effector, which is equal in magnitude to the sensed contact force but opposite in direction.

Given such a simulator, the world model identification problem is formulated as solving the following optimization:

$$\begin{aligned} \underset{\theta}{\mathbf{argmin}} \sum_{t=t_1}^{t_T} \mathcal{L}\Big(\hat{o}^t\big(q^t(\theta), \theta\big), o^t\Big) \\ \mathbf{s.t.} \quad q^{i+1}, \dot{q}^{i+1} = g(q^i, \dot{q}^i, u^i, \theta) \quad \forall t = 1, \cdots, T-1. \end{aligned} \tag{1}$$

The optimization is solved over a physical motion sequence of T timesteps, with the objective function $\mathcal{L}$ encourages the simulated observation $\hat{o}^t\big(q^t(\theta), \theta\big)$ to match the ground-truth observation o^t.

IV. Method

In this section, we first detail the object representation, which is key to our method. Then we describe the differentiable simulator and details on solving the optimization described in Eqn. 1.

A. Differentiable Object Representation

An ideal object representation for world model identification needs to be flexible to allow learning of complex object geometries, topologies, and appearance properties while being compatible with rigid body simulators for collision detection. Topology-agnostic geometries such as point clouds [19] and GS [9] do not allow one to calculate the penetration depth between bodies. On the other hand, meshes [15] do not allow large geometric and topological changes.

We find that the SaP framework [11], when augmented by additional appearance properties poses an ideal representation for our purpose. Briefly, this framework represents the object using a point cloud with normals on the object surface, denoted as $\mathcal{P} = \{(p \in \mathbb{R}^3, n \in \mathbb{R}^3)\}$. These normal directions induce a discrete vector field $v(x) = \sum_{\langle p,n \rangle \in \mathcal{P}} n \mathbb{I}[x = p]$. SaP then uses Poisson reconstruction [26] to recover an underlying implicit indicator field $\chi(x)$ that describes the occupancy of the solid geometry, i.e. whether x is inside or outside the geometry, and matches its gradient field with $v(x)$ by solving the variational problem:

$$\underset{\chi}{\mathbf{argmin}} \int_\Omega \|\nabla \chi(x) - v(x)\|^2,$$

which amounts to solving the Poisson equation $\Delta \chi = \nabla \dot{v}$. SaP discretizes the indicator field χ on a uniform grid domain Ω, which allows the efficient solution of χ via GPU-accelerated Fast Fourier Transform (FFT) with well-defined derivatives. We use a $128 \times 128 \times 128$ discretized grid χ for all the experiments in this paper.

The indicator field χ is then transformed to a triangle mesh $\mathcal{M}$ with a differentiable marching cubes algorithm [12]. Collision detection can then be easily achieved with standard techniques for meshes. To enable appearance modeling, we further augment with a grid of appearance properties, with the same grid resolution as the one storing the indicator field χ. The appearance property is then propagated to the mesh vertices via tri-linear interpolation. In this work, we only store and render the color field, denoted as ψ, but other appearance properties can be incorporated in the same manner as required by more advanced differentiable rendering equations. The mesh can then be rendered using any differentiable renderer framework such as [27], [28], for which we use the open source implementation in PyTorch3D [29]. Specifically, at the time instance t, we invoke the renderer with the object transformed to q^t and the camera transformed to c^t. Our parametrization of the object's physics and appearance is defined as:

$$\theta \triangleq \langle M(q^i), \mu, \mathcal{P}, \psi \rangle,$$

where the first two parameters are mass-inertial properties and frictional coefficients, and the last two parameters are the oriented point cloud for SaP and color field.

For the terrain, we simply use an oriented and colored point cloud $\mathcal{P}_t$ to represent the terrain as we do not need to simulate interactions between 2 terrains. The terrain is rendered from the colored point cloud with an alpha compositor [30] also using the PyTorch3D library and we set the radius of each point to be 0.015 m.

B. Differentiable Simulator

For our application, we only consider unconstrained rigid body dynamics with dry frictional contacts. Note that additional physical constraints for describing objects such as soft bodies and articulated objects can be potentially incorporated into our framework and its differentiation has been well-studied, e.g. in [16].

The governing equation of motion for rigid bodies and the time discretization method are well-established, and we refer the readers to Anitescu et al. [31] for details. The equation is summarized as follows:

$$M(q^i)\ddot{q}^i = C(q^i, \dot{q}^i) + J^i u^i + J^\perp \tau^\perp + J^\parallel \tau^\parallel, \tag{2}$$

with $M(q^i)$ being the generalized mass matrix, $C(q^i, \dot{q}^i)$ being the centrifugal, Coriolis, and gravitational force, $J^i, J^\perp, J^\parallel$ being the Jacobian matrix for the external, normal, and tangent contact forces at all the detected contact points, respectively. Finally, $\tau^\perp, \tau^\parallel$ are the contact forces. At each time step, a mixed linear complementarity problem (LCP) is solved to calculate the constraint forces $\tau^\perp, \tau^\parallel$, yielding the final acceleration $\ddot{q}^i$, and we then integrate the configuration forward in time [32], [33] as:

$$\dot{q}^{i+1} = \dot{q}^i + \ddot{q}^i \delta t \quad q^{i+1} = q^i + \dot{q}^i \delta t, \tag{3}$$

Fig. 3. The experiment setups for the simulation (left) and physical (right) experiments. 9 objects are used for simulation with the PyBullet simulator, including 8 YCB objects and a green box. For the real-world experiments, three YCB objects (`Drill`, `Mustard`, and `Sugar`) are used. A UR5e arm equipped with a pusher and a ATI Gamma F/T sensor and an overhead Realsense D435 RGB-D camera are used. Note that only the circled object in the real-world setup is the object of interest and everything else is treated as the static terrain.

with δt being the timestep size. The mixed LCP problem is formulated as:

$$\begin{cases} 0 \le \tau^\perp & \perp J^{\perp T} \dot{q}^{i+1} \ge 0 \\ 0 \le \tau^\parallel & \perp \lambda e + J^{\parallel T} \dot{q}^{i+1} \ge 0 \\ 0 \le \lambda & \perp \mu \tau^\perp - e^T \tau^\parallel \ge 0, \end{cases} \tag{4}$$

with e being the unit vector and μ being the frictional coefficient. λ is an auxiliary variable encoding the stick or slip frictional state. To differentiate through the simulator, we adopt the differentiation technique proposed by [14], [13], where the result of the LCP is made differentiable by solving with a primal-dual method and performing sensitivity analysis at the solution to yield derivatives with respect to the problem data. In this way, the derivatives propagate the gradient information to the Jacobian matrix $J^{\perp,\parallel}$, and finally to the object geometric parameters $\mathcal{P}$. In summary, Eqn. 2,3,4 defines our differentiable simulator function g. In particular, we adopt the differentiable simulator proposed by Strecke et al. [13] for its fast implementation on GPU.

C. World Model Identification

Even with our jointly differentiable physical and appearance models, solving Eqn. 1 can still be rather challenging. This is mainly because our initial guess can be very poor, especially in the occluded region. As a result, the naïve gradient descent method can take many iterations and is prone to converging to poor local minima. To mitigate this, we use two stages of optimization, and we further leverage 3D foundation models trained on web-scale data to generate reasonable initial guesses of the rigid object in the scene from partial visual observations.

1) Two-stage Optimization: We note that while the initial guess can deviate significantly from our observations, deviations in geometry and appearance can be largely corrected by considering only the first observation, i.e. $\langle t_1, e^{t_1}, f^{t_1}, o^{t_1}, c^{t_1}, m^{t_1} \rangle$. Therefore, our first stage considers

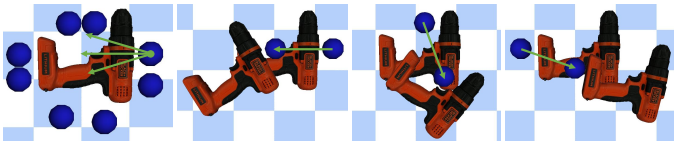


Fig. 4. The pushing trajectories used in the experiments. Left: The 8 starting locations of the floating spherical robot pushing trajectories and 3 pushing directions towards the robot at one of the starting locations for the `Drill` object in the simulation experiments. Middle and right: the training trajectory and 2 sample testing trajectories, with the first and last frames shown. [Best viewed in color.]

only the first time instance and optimizes θ using the following loss:

$$\begin{aligned}\mathcal{L}(\hat{o}^{t_1}, o^{t_1}) =& c_1\mathcal{L}_{\text{rgb}}(\hat{o}^{t_1}, o^{t_1}) + c_2\mathcal{L}_{\text{depth}}(\hat{o}^{t_1}, o^{t_1}) + \\ & c_3\mathcal{L}_{\text{pcd}}(o^{t_1}, \theta) + c_4\mathcal{L}_{\text{pen}}(\theta) + c_5\mathcal{L}_{\text{balance}}(\theta) + \\ & c_6\mathcal{L}_{\text{reg}}(\theta, \theta_0) + c_7\mathcal{L}_{\text{smooth}}(\mathcal{P}),\end{aligned}$$

with $(c_1, \ldots, c_7)$ denoting weight terms. Here, $\hat{o}^{t_1}$ is the rendered RGB-D image at t_1. $\mathcal{L}_{\text{rgb}}$ is a loss on the RGB images, defined as a weighted sum of l_1 distance and D-SSIM terms: $\mathcal{L}_{\text{rgb}} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{SSIM}}$, where we set $\lambda = 0.2$. $\mathcal{L}_{\text{depth}}$ is the l_1 distance on the depth images. When calculating the three loss terms $\mathcal{L}_{1,\text{SSIM,depth}}$, the robot is masked out according to the segmentation mask m^{t_1}. $\mathcal{L}_{\text{pcd}}$ is a unilateral Chamfer distance between the point cloud generated from the observed RGB-D image pixels belonging to the object and the mesh vertices generated from $\mathcal{P}$, which is defined as the average of the minimum distance between each point on the observed point cloud and any mesh vertex. The use of a unilateral Chamfer distance as opposed to the regular Chamfer distance is necessary since the observed object point cloud only includes points on the object surfaces visible from the camera viewpoint. The terms $\mathcal{L}_{\text{pen}}$ and $\mathcal{L}_{\text{balance}}$ encourage the object to achieve static force equilibrium while having no penetrations. $\mathcal{L}_{\text{pen}}$ penalizes the object mesh penetrations in the terrain, and $\mathcal{L}_{\text{balance}} = \sum_1^k \|p_o^{i+t_1} - p_o^{t_1}\|_1$ is the sum of the positional changes of the object from the initial position $p_o^{t_1}$ over k steps by simulating forward with no robot actions. In the case where the initial guess of the object geometry does not come in contact with the terrain at t_1, $\mathcal{L}_{\text{balance}}$ allows the computation of the gradient information to expand the geometry towards the terrain once the object falls due to gravity and contacts the terrain during the k steps. We use $k = 3$ for all the experiments in this paper. These two terms provide a strong hint for the occluded part of the object. For example, when an object is lying on the table, our RGB-D observation will not cover the bottom of the object. However, our model will guide SaP to fill the bottom-side geometries by encouraging the object to settle on the table. Finally, the last two terms regularize the object shape, where L_{reg} is the L_2 norm between the SaP points and the initial SaP points and L_{smooth} is the Laplacian smoothing objective on the object mesh.

In addition, the use of both $\mathcal{L}_{\text{depth}}$ and $\mathcal{L}_{\text{pcd}}$ is necessary. When only $\mathcal{L}_{\text{depth}}$ is used, if the estimated mesh is smaller than the ground-truth object geometry, the predicted depth pixels that are supposed to reach the ground-truth mesh do not hit the estimated mesh, and there is no gradient information for expanding the geometry. Similarly, $\mathcal{L}_{\text{pcd}}$ does not inform the SaP to not expand over the observed point cloud, and $\mathcal{L}_{\text{depth}}$ prevents the object geometries from occupying the supposed background.

After the first stage, we have tuned our model to match the first observation. When we move on to the second stage, we incorporate all timesteps by applying the robot controls. To make sure the geometry and appearance of the object stay close to that in the first time step, we still use the loss from the first stage on the first time frame, except for the penetration and balance losses. For the rest of the time frames, we use the loss $\mathcal{L} = c_8\mathcal{L}_{\text{pcd}} + c_9\mathcal{L}_{\text{robot}}$, where the first term has the same definition as in the first stage and $\mathcal{L}_{\text{robot}}$ is the squared distance between the ground-truth and predicted robot end-effector positions. To calculate this loss, we apply the robot control forces from the initial state, integrate forward in time, and render two intermediate and the last frames, instead of every time frame for computational efficiency. During our optimization, the chain of gradients is back-propagated through the following recursive rule:

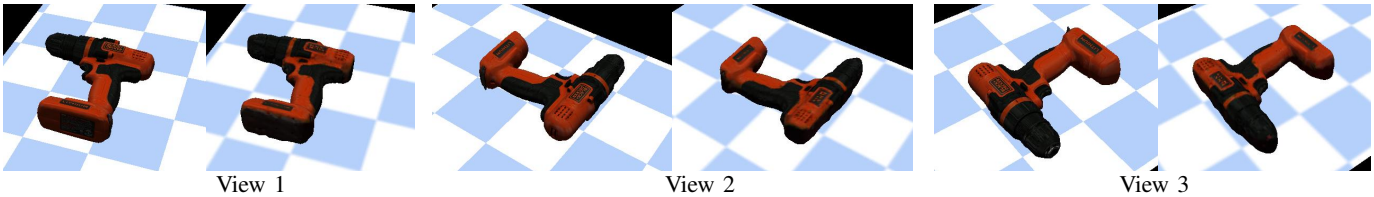
$$\frac{d\mathcal{L}(\hat{o}^{t_i}, o^{t_i})}{d\theta} = \frac{\partial\mathcal{L}}{\partial\theta} + \frac{\partial\mathcal{L}}{\partial q^i}\left[\frac{\partial q^i}{\partial\theta} + \frac{\partial q^i}{\partial q^{i-1}}\frac{dq^{i-1}}{d\theta}\right],$$

where the first two terms $\partial\mathcal{L}/\partial\theta, \partial\mathcal{L}/\partial q^i$ is the derivatives of the rendering equation, and the remaining terms in the bracket are the derivatives of the simulator.

2) Geometry Prior: Our method relies on a reasonable initial guess. Imagine the case with an object settling on the edge of a table and the camera does not observe the contact between the two. The initial guess of the occluded part of the object could be very short and cause the object to directly fall down without touching the table. This cannot be recovered by our optimization since the object never hits the table and there are no gradients for correcting the geometries. To obtain a reasonable initial guess of the geometries and appearance of the rigid object of interest from partial visual observations, we take advantage of large reconstruction models [34] that predict object 3D models from a single RGB image, trained on web-scale data. In particular, we use TripoSR [35] in our experiments with the segmented RGB image of the object as the input image. Since the generated mesh is scale- and transform-agnostic, we apply RANSAC and the scale-aware iterative closest point algorithms with Open3D [36] to register the mesh to the partial object point cloud, computed from the RGB-D image at the first time instance.

Finally, in all the experiments of this work, we assume that the occluded terrain by the object is flat, and complete the terrain by fitting a plane of points, where the colors match the nearest visible points of the terrain. In addition, in all the experiments, we do not optimize the point cloud position of the terrain and optimize only the colors. Although these settings are simplifying, we believe a similar approach could be adopted that predicts the geometry of the occluded rigid terrain from a geometry prior model and optimizes for the terrain geometry simultaneously, although more online data may be required to resolve the ambiguities of the contacts between two occluded geometries.

Fig. 5. The predicted and ground-truth poses of the 5 different objects at the end of sampled testing trajectories for the simulation experiments. After training, the predicted poses are obtained by applying the control forces from the initial pose and integrating forward in time. The predicted object poses are highlighted with a yellow silhouette and overlaid with the ground-truth object, blue floating spherical robot, and background. [Best viewed in color.]



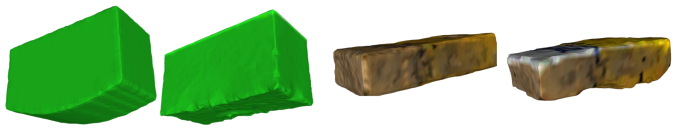
View 1 View 2 View 3

Fig. 6. The ground-truth (left) and predicted (right) RGB images of 3 novel views of the `Drill` in simulation. The optimized mesh shape and geometry match the ground truth well, although lacking the fine details that can not be observed from the top view. The terrain checkers are not as sharp as the ground truth due to the use of point rendering of the colored terrain point cloud.

	Dynamics Parameter Error		Trajectory Prediction Error			
Method	mass (kg)	μ	Unilateral Chamfer (mm)	Pos. (mm)	Rot. (°)	Trans. Vel. (m/s^2)
Ours	0.0728	0.106	8.69	15.5	16.7	0.0351
PhysGS [25]	0.225	0.400	24.2	42.8	31.8	0.436

TABLE I
AVERAGE DYNAMICS PARAMETER IDENTIFICATION AND NOVEL TRAJECTORY PREDICTION ERRORS FOR ALL OBJECTS IN THE SIMULATION EXPERIMENTS

Fig. 7. Results of three example testing trajectory of the physical experiments. The predicted object and robot poses with the optimized θ highlighted with a yellow silhouette are overlaid with the ground-truth object, robot, and background. [Best viewed in color.]



Initial Optimized Initial Optimized

Fig. 8. The initial guess and optimized shape of the `Box` object in simulated experiments and the `Sugar` object in physical experiments during stage 1. The algorithm is able to correct the bulging on the underside of the `Box` that would intersect the terrain. On the other hand, the initial mesh of `Sugar` is too thin and does not touch the terrain. Our algorithm is able to optimize the shape so that it satisfies physics constraints.

V. EXPERIMENTS AND RESULTS

To validate our method, we first conduct experiments with simulated data, and then in the real world.

A. Simulation Experiments

Shown in Fig. 3, we conduct all the simulated experiments using data collected with the PyBullet simulator [37]. We use a simple green box object (`Box`) and 8 objects (`Gelatin`, `RubiksCube`, `Spam`, `TunaCan`, `Mustard`, `Bleach`, `Drill`, and `Sugar`) from the YCB object dataset [38], which covers diverse shapes, sizes, and textures. The objects are placed on a flat surface with checker patterns and pushed by a floating sphere robot, while a static overhead camera takes pictures. As shown in Fig. 4, 24 pushing trajectories are adopted, where one is used to optimize the world model and the rest for evaluating the optimized model. To make sure that the pushes are diverse, we pick 2 starting locations on each of the four sides of the object, and push in three directions that are 20° apart at each of these 8 locations toward the object. Similar pushes are used for all the other objects where the starting locations are adjusted based on the size of the objects. The trajectories push the objects up to 12 cm and 80°. All the trajectories have $T = 30$ time steps with $\delta t = 0.01$ s. We use the following weights for optimization: $[c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9] = [10, 500, 2000, 100, 100, 100, 4000, 500, 100]$. These terms are not carefully tuned and are set such that each term has a similar order of magnitude at the start of optimization for our experiments. The physical parameters we optimize for include the mass, surface coefficient of friction, the center of mass, and the rotational inertial. The center of mass is initialized at the geometry center of the initial shape guess. The rotational inertia is initialized by treating the object as a box, whose dimensions are the bounding box of the initial geometry guess. We assume that the rotational inertia only has diagonal terms. For all experiments including simulations and real-world experiments, the surface coefficient of friction is initialized at 0.2 and the mass is initialized at 0.2 kg.

We first evaluate whether our method can identify the shape and physical parameters accurately, such that it generalizes well to new physical interactions. We compare our method against a recently proposed method that represents the world jointly with Gaussian splats and physical particles and allows it to perform both novel-view rendering and physics-based trajectory predictions [25]. We refer to this method as PhysGS. This method performs simulation with a particle-based simulator [39]. In the original paper, the physical parameters of the particles are arbitrarily set, and virtual forces are optimized to match the observations and predictions on the training trajectory. To allow more accurate prediction for new trajectories, we optimize the total mass and coefficient of friction of the object particles with grid search using the partial Chamfer distance between the observed object point cloud and the predicted object particles on the last time frame in the trajectory.

We report the quantitative results for dynamics parameter estimation and novel trajectory predictions for all objects in the simulation experiments in Table I, which are the average across all objects. The dynamics parameter estimation error from the training trajectory, the average pose error, the unilateral Chamfer distance, and the translational velocity error at the end of the testing trajectories are reported. We also show some qualitative examples that are representative of the average errors in Fig. 5. The final pose and velocity of the objects are obtained by applying the control forces and integrating forward in time. Our method identifies the dynamics parameters accurately and shows low trajectory prediction errors. We would like to point out in particular that the average rotation error is heavily skewed by the `TunaCan` object since our method currently uses the object surface point cloud to track the object during stage 2 optimization, and cannot properly differentiate the rotation of a cylindrical object. As a result, the average rotation error is 28.0° for `TunaCan`, and we aim to address this issue in future work. On the other hand, PhysGS generalizes very poorly to the testing trajectories. While this is partially because of the lack of proper dynamics parameters and physics-based shape estimation, we also find that the particle-based simulator is extremely sensitive to simulator parameters and have poor physical fidelity, especially for rigid objects. We also show an example of the initial and optimized geometries of the `Box` object in Fig. 8 (left), which demonstrate the ability of our method to adjust occluded geometry based on the physics. Next, we also evaluate the quality of the novel-view synthesis of our method. For each of the testing objects, we evaluate the synthesized RGB images from 10 novel viewpoints around the scene, and our method achieves 0.00225 of mean squared error (MSE), 0.965 of structural similarity index measure (SSIM) and 26.5 of peak signal-to-noise ratio (PSNR). We also show some examples of novel view synthesis of the `Drill` object in Fig. 6, which matches the ground truth very well.

B. Physical Experiments

Shown in Fig. 3, we conduct physical experiments with a UR5e robot arm equipped with an ATI Gamma F/T sensor and a pusher with a semispherical end, and a static overhead RealSense D435 RGB-D camera. We use similar pushing trajectories to those for the simulation experiments, but only use 6 trajectories with two different starting locations from one side of the object. We then use one trajectory for training and the rest for evaluation with 3 YCB objects: power drill (`Drill`), sugar box (`Sugar`), and mustard bottle (`Mustard`). We use a total of $T = 48$ time steps with $\delta t = 0.03$ s. We use the same optimization settings as the simulated experiments.

On average across all testing trajectories and testing objects, our method achieves a mass identification error of 0.186 kg and 6.10 mm of unilateral chamfer distance between the observed object point cloud and the predicted object at the last frame of the trajectory. Note that we only reported the mass identification error because the ground truth is very easy to measure while measuring the surface friction requires a specialized setup. Since we do not have access to the ground-truth object poses and only have access to the raw observations, we report the unilateral Chamfer distances between the observed object point cloud and the predicted object geometry. The train and test prediction results for three sample trajectories are visualized in Fig. 7. Overall, the errors are comparable to those from the simulated experiments. In addition, we show the initial and optimized shape of the `Sugar` object in Fig. 8 on the right, where the initial shape is too thin and does not contact the terrain below. Our algorithm modifies the occluded geometry to satisfy the physics.

VI. Limitations

Our method assumes ground-truth object masks from the scene, which might not always be possible even with advanced foundational segmentation models. In addition, our method does not consider more advanced appearance models, lighting sources, and shadows. As a result, the rendered scenes could have artifacts that do not match the real-world observations. Currently, each optimization run is completed in under 15 mins on a standard PC with an Intel i9-13900KF CPU, 64 GB of RAM, and a GeForce RTX 4090 GPU. While this is not ideal for online robotics applications, we intend to reduce the runtime by using better initial guesses of geometry and physical parameters from data-driven pre-trained models and more efficient implementation. Finally, our method currently struggles on objects whose rotation can not be properly identified from a surface point cloud, such as a cylinder. We aim to explore tracking methods that also leverages surface textures for pose tracking in future work.

VII. Conclusion

We propose a novel algorithm to solve the task of identifying objects' physical properties as well as the geometry and appearance, a crucial step in downstream robot manipulation tasks. To the best of our knowledge, this is the first method that allows the joint optimization of all of these properties. Our method combines the merit of SaP object representation [12], differentiable collision detection [40], and differentiable simulation [13]. Although our method has several limitations, it opens doors to a rich spectrum of future research topics.

Some potential future directions include extending our method to identify multi-body dynamic systems with additional constraints, more advanced appearance models, and physics-based perception to correct for wrong object masks.

REFERENCES

[1] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[2] J. Bruce, M. D. Dennis, A. Edwards, J. Parker-Holder, Y. Shi, E. Hughes, M. Lai, A. Mavalankar, R. Steigerwald, C. Apps *et al.*, "Genie: Generative interactive environments," 2024.

[3] F. Zhu, H. Wu, S. Guo, Y. Liu, C. Cheang, and T. Kong, "Irasim: Learning interactive real-robot action simulators," *arXiv preprint arXiv:2406.14540*, 2024.

[4] M. Yang, Y. Du, K. Ghasemipour, J. Tompson, L. Kaelbling, D. Schuurmans, P. Abbeel, U. Berkeley, and G. DeepMind, "Learning interactive real-world simulators," *International Conference on Learning Representations*, 2024.

[5] C. Song and A. Boularias, "Learning to Slide Unknown Objects with Differentiable Physics Simulations," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[6] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, "Differentiable simulation for physical system identification," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3413–3420, 2021.

[7] K. Ehsani, S. Tulsiani, S. Gupta, A. Farhadi, and A. Gupta, "Use the force, luke! learning to predict physical forces by simulating effects," *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[8] 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," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[9] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering." *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139–1, 2023.

[10] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *IEEE international conference on robotics and automation*, 2015.

[11] S. Peng, C. Jiang, Y. Liao, M. Niemeyer, M. Pollefeys, and A. Geiger, "Shape as points: A differentiable poisson solver," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13 032–13 044, 2021.

[12] Y. Liao, S. Donne, and A. Geiger, "Deep marching cubes: Learning explicit surface representations," in *IEEE conference on computer vision and pattern recognition*, 2018.

[13] M. Strecke and J. Stueckler, "DiffSDFSim: Differentiable rigid-body dynamics with implicit shapes," in *International Conference on 3D Vision (3DV)*, Dec. 2021.

[14] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and Feature-Complete Differentiable Physics Engine for Articulated Rigid Bodies with Contact Constraints," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[15] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal, "An End-to-End Differentiable Framework for Contact-Aware Robot Design," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[16] Y. Qiao, J. Liang, V. Koltun, and M. Lin, "Differentiable simulation of soft multi-body systems," *Advances in Neural Information Processing Systems*, 2021.

[17] T. A. Howell, S. L. Cleac'h, J. Brüdigam, J. Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable physics engine for robotics," *arXiv preprint arXiv:2203.00806*, 2022.

[18] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "Add: Analytically differentiable dynamics for multi-body systems with frictional contact," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–15, 2020.

[19] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3d point clouds," in *International conference on machine learning*. PMLR, 2018, pp. 40–49.

[20] B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi, "Cvxnet: Learnable convex decomposition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 31–44.

[21] S. Pfrommer, M. Halm, and M. Posa, "Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations," in *Conference on Robot Learning*. PMLR, 2021, pp. 2279–2291.

[22] N. Sharp and A. Jacobson, "Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–16, 2022.

[23] T. Xie, Z. Zong, Y. Qiu, X. Li, Y. Feng, Y. Yang, and C. Jiang, "Physgaussian: Physics-integrated 3d gaussians for generative dynamics," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 4389–4398.

[24] K. J. Åström and P. Eykhoff, "System identification—a survey," *Automatica*, vol. 7, no. 2, pp. 123–162, 1971.

[25] J. Abou-Chakra, K. Rana, F. Dayoub, and N. Suenderhauf, "Physically embodied gaussian splatting: A visually learnt and physically grounded 3d representation for robotics," in *8th Annual Conference on Robot Learning*, 2024.

[26] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, no. 4, 2006.

[27] S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3d reasoning," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 7708–7717.

[28] S. Laine, J. Hellsten, T. Karras, Y. Seol, J. Lehtinen, and T. Aila, "Modular primitives for high-performance differentiable rendering," *ACM Transactions on Graphics*, vol. 39, no. 6, 2020.

[29] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, "Accelerating 3d deep learning with pytorch3d," *arXiv:2007.08501*, 2020.

[30] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson, "Synsin: End-to-end view synthesis from a single image," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 7467–7477.

[31] M. Anitescu and F. A. Potra, "Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems," *Nonlinear Dynamics*, vol. 14, pp. 231–247, 1997.

[32] M. B. Cline, "Rigid body simulation with contact and constraints," Ph.D. dissertation, University of British Columbia, 2002.

[33] D. E. Stewart and J. C. Trinkle, "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction," *International Journal for Numerical Methods in Engineering*, vol. 39, no. 15, pp. 2673–2691, 1996.

[34] Y. Hong, K. Zhang, J. Gu, S. Bi, Y. Zhou, D. Liu, F. Liu, K. Sunkavalli, T. Bui, and H. Tan, "Lrm: Large reconstruction model for single image to 3d," *arXiv preprint arXiv:2311.04400*, 2023.

[35] D. Tochilkin, D. Pankratz, Z. Liu, Z. Huang, , A. Letts, Y. Li, D. Liang, C. Laforte, V. Jampani, and Y.-P. Cao, "Triposr: Fast 3d object reconstruction from a single image," *arXiv:2403.02151*, 2024.

[36] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[37] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[38] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics and Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[39] M. Macklin, "Warp: A high-performance python framework for gpu simulation and graphics," 2022.

[40] E. Guendelman, R. Bridson, and R. Fedkiw, "Nonconvex rigid bodies with stacking," *ACM transactions on graphics (TOG)*, vol. 22, no. 3, pp. 871–878, 2003.