Learning a Particle Dynamics Model with Real-world Videos

Chanho Kim Suhas V. Sumukh Li Fuxin
Oregon State University
{kimchanh, suhas.sumukh, fuxin.li}@oregonstate.edu

Abstract

Data-driven learning approaches for physics simulation, sometimes referred to as world models, have emerged as promising alternatives to traditional physics simulators due to their differentiable nature. Prior work has demonstrated impressive results in predicting the motions of rigid and non-rigid objects in complex scenes involving multiple interacting bodies. However, these models are typically trained in simulated environments because obtaining perfect state information such as complete scene point clouds and point correspondences over time is challenging in real-world settings. This reliance on synthetic data can limit their applicability when the sim-to-real gap is large. In this work, we aim to overcome these limitations by introducing a novel framework for training neural object dynamics models directly from unlabeled real-world videos. Specifically, we propose to learn a particle-based dynamics model compatible with a Gaussian splatting framework, which operates on dense particles derived from Gaussians (i.e., particles with scales and rotations) and predicts their position and rotation changes over time. The model is trained via rendering supervision, enabling learning from real-world videos without requiring particle-level labeled states. Our model operates directly on dense Gaussians without relying on heuristic subsampling anchor points. To enable this study, we also present a real-world dataset consisting of about 500 videos capturing diverse object interactions. The dataset and code will be publicly released at our project webpage: https://chkim403.github.io/gs_physics.

1. Introduction

Humans develop an intuitive understanding of physics early in life simply by observing the world, long before studying the physics subject in any school. This innate ability enables us to effortlessly track the motion of nearby objects and make reasonably accurate predictions about their future trajectories based on past movements. Such skills are essential for acting safely and effectively in a complex environment where constant interactions with other objects and people are inevitable.

Endowing AI agents with similar physical reasoning capabilities could benefit numerous applications such as manufacturing, robotics, and generative modeling [1, 9, 18, 36]. One approach to achieve this is through classical particle-based simulations, such as the Material Point Method (MPM) [8, 34]. However, these methods typically require all physical parameters of the particles to be predefined, an assumption that rarely holds in real-world scenarios.

Alternatively, learning particle dynamics directly using neural networks has attracted considerable attention due to its efficiency and differentiability, which makes such models more suitable as components of larger end-to-end trainable systems. Despite this promise, existing models remain limited in several ways. Many methods [6, 12, 14, 19, 21] are trained primarily on simulated data, as obtaining real-world 3D labeled data, such as dense point cloud correspondences across frames, is expensive and challenging, particularly for videos. Even when trained on real-world sequences [22], the supervision signals are often noisy because the loss relies on approximate distance functions such as the Chamfer distance. This lack of reliable supervision limits the applicability of current models in diverse real-world settings.

Recent advances in differentiable rendering such as Gaussian Splatting (GS) and Neural Radiance Fields (NeRF) [11, 17] present a promising alternative. By enabling gradients to flow from 2D observations back to 3D, these methods allow 3D models to be **supervised directly from images**. This opens the door to learning dynamics from real videos without ground-truth 3D labels or a physics simulator, where there are **much more** data available online to scale training. Yet prior work in this direction has focused almost exclusively on single-object scenes [4, 15, 36, 37], hence does not capture the complex discontinuous interactions required for multi-object collision dynamics.

In this work, we take a step on learning **multi-object collision dynamics** directly from **real world videos** via rendering-based supervision, a capability that was previously unavailable to the community to the best of our

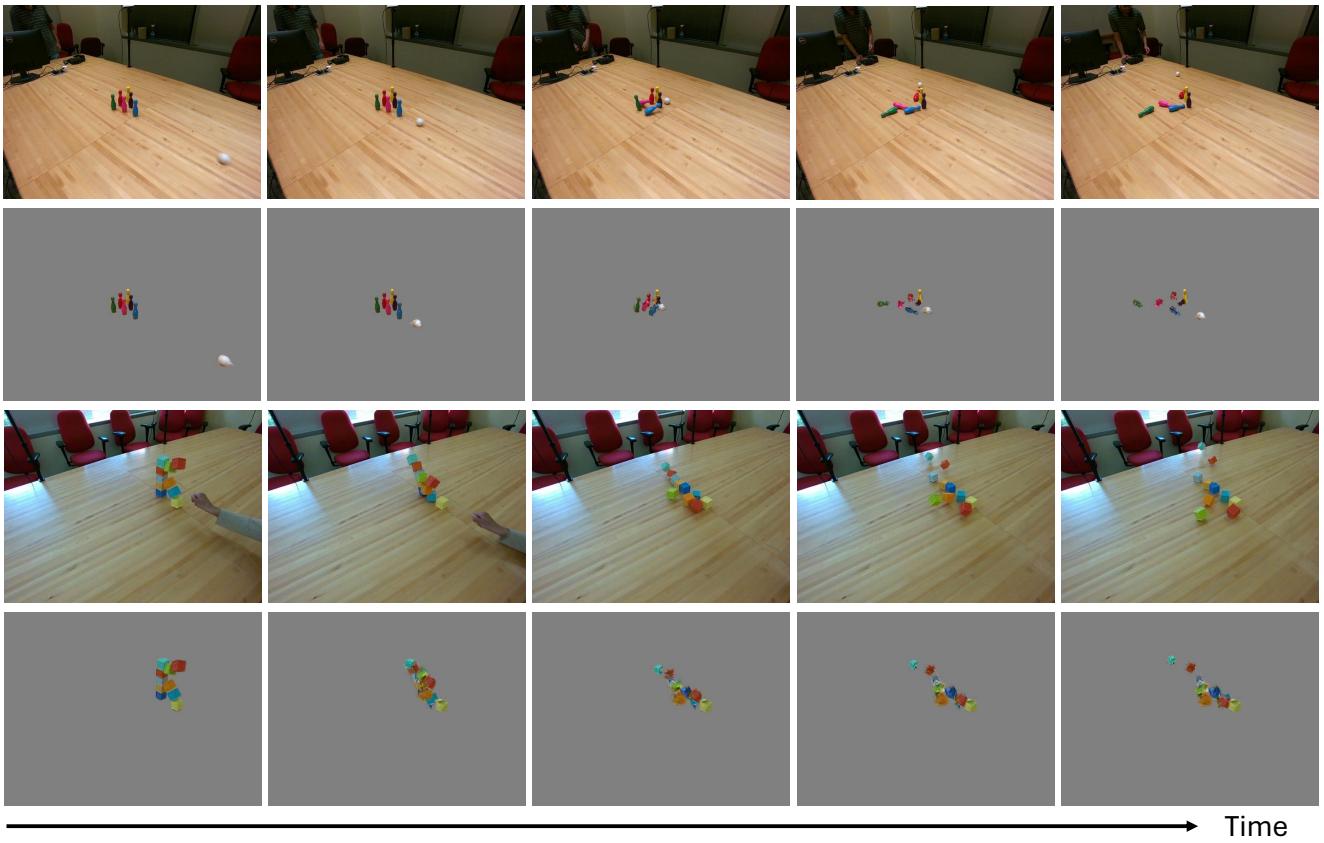

Figure 1. Example sequences illustrating the physical scenarios of interest. The dataset captures multi-object interactions with complex collision dynamics, including bowling-style impacts and cube-stack collapses. The first and third rows show the original RGB sequences, while the second and fourth rows show renderings from the 3D Gaussian rollouts predicted by our dynamics model for the same viewpoints. Our approach enables training dynamics models in multi-object settings directly from real-world videos.

knowledge. We propose a model that learns the collision dynamics of multiple interacting objects using only 2D object masks from a SAM-style video segmentation model [20, 25], combined with a differentiable rendering loss from multiple calibrated viewpoints. Learning in this setting introduces several challenges absent in prior work: 3D trajectory extraction from partial 2D cues, assigning Gaussians to different objects through occlusions and collisions, and predicting future Gaussian states in a feed-forward manner.

To support this problem, we also collect a new real-world multi-view dataset (Fig. 1) containing about 500 videos captured from four calibrated cameras across two challenging scenarios—falling cube stacks and bowling—that exhibit rich, nontrivial multi-object interactions rarely available in prior datasets.

In summary, our contributions are:

- We presented the first approach to extend multi-object, action-free 3D particle dynamics models to training on real videos with rendering loss without knowledge of the physical properties of the particles.
- We capture around 500 multi-view videos with complex contact events that serve as a benchmark for learning real-world physical interactions.
- We provide reference designs and ablations for key system components, including Gaussian trajectory initialization, assigning Gaussians to objects and multi-step rollout training. Code and dataset will be released.

2. Related Work

Recent advances in rendering-based 3D reconstruction such as Gaussian Splatting (GS) and Neural Radiance Fields (NeRF) [11, 17] have opened new possibilities for leveraging 2D image observations as ground-truth supervision signals for 3D models. Several prior works have leveraged NeRF to train GNN-based dynamics models via rendering supervision [4, 30, 35]. However, NeRF is inherently a volumetric model and can only model particle dynamics through an implicit deformation field, which can be more ill-posed than necessary and it is difficult to add a notion of object among particles or impose rigidity constraints.

Gaussian Splatting (GS) [11] offers a more explicit scene representation, making it possible to use well-established architectures such as GNNs and point-cloud backbones to process 3D Gaussians directly. Recent work in robotics [15, 36, 37] has explored training world models directly from real world videos, with the GS framework either enabling rendering-based supervision [15, 37] or enabling test-time video generation [36]. However, these approaches primarily model the effects of robot actions on a single manipulated object, limiting their applicability to specific task contexts.

For multi-object collision scenarios, rendering-based supervision has also been used to learn dynamics models for complex collisions across multiple interacting objects [2, 26, 29, 39], but prior work has been limited mostly to simulated data, where rendering-based loss makes less sense due to the availability of better ground truth. [2] demonstrated learning collision dynamics from real world labeled videos, but their dataset contained only simple scenes involving a single cube being thrown onto a table, and the ground truth cube poses were obtained using visual markers placed on the cube. To our knowledge, our work is the first to learn non-trivial multi-object collision dynamics directly from real-world videos without markers.

Gaussian Splatting has also been used to generate simulatable assets, where Gaussians extracted from real-world observations are augmented with object properties. These properties are either user-specified [34] or estimated from video observations [10, 40]. The resulting simulatable Gaussians can then be used to produce rollouts under arbitrary forces through MPM or Euler integration. In contrast to these approaches, we aim to learn a dynamics model that generates rollouts directly, without any manual tuning or additional system identification steps at test time.

3. Method

We base our approach on prior work [12] that utilizes a point cloud backbone [32, 33] for object interaction modeling with point cloud input. A set of 3D Gaussians produced by Gaussian Splatting is essentially a 3D point cloud where each point is associated with additional attributes such as scale and rotation. Therefore, in this work we directly treat Gaussians as points and apply a point cloud backbone.

In Sec. 3.1–3.2, we introduce our problem setup. In Sec. 3.3, we briefly review the prior work that forms the basis of our convolution-based interaction modeling. In Sec. 3.4–3.6, we present our approach for enabling GS-based collision dynamics learning.

3.1. Gaussian Representation

We start with a set of Gaussians that are produced by an external Gaussian Splatting algorithm [11] on a single frame. Each Gaussian i is associated with its center $\mathbf{x}^{(i)} \in \mathbb{R}^3$, rotation $\mathbf{R}^{(i)} \in SO(3)$, scale $\mathbf{s}^{(i)} \in \mathbb{R}^3$, color $\mathbf{c}^{(i)} \in \mathbb{R}^3$, and opacity $o^{(i)} \in \mathbb{R}$:

$$G^{(i)} = \left(\mathbf{x}^{(i)},\ \mathbf{R}^{(i)},\ \mathbf{s}^{(i)},\ \mathbf{c}^{(i)}, o^{(i)}\right). \tag{1}$$

Gaussian Splatting allows these Gaussians to be rendered into images via a differentiable Gaussian renderer, which enables rendering-based supervision.

The rendering equation for a pixel I is:

$$I(u) = \sum_{j=1}^{J} \mathbf{c}^{(j)} \alpha_c^{(j)}(u) \prod_{k=1}^{j-1} (1 - \alpha_c^{(k)}(u)) \tag{2}$$

where all the J Gaussians are sorted by the distance from their centers to the camera, and $\alpha_c^j(u)$ is the opacity $o^{(j)}$ multiplied with the projection of the Gaussian $G^{(j)}$ density along the camera ray that passes through u.

Let the three input frames be at times $t-2, t-1, t$. For each Gaussian i in frame t, we augment its representation with its two most recent velocities

$$\mathbf{v}_{t-1}^{(i)} = \mathbf{x}_{t-1}^{(i)} - \mathbf{x}_{t-2}^{(i)}, \qquad \mathbf{v}_t^{(i)} = \mathbf{x}_t^{(i)} - \mathbf{x}_{t-1}^{(i)}. \tag{3}$$

which can be obtained from any video via point tracking approaches such as [7] (see Sec. 4.2 for more details). We choose this representation which does not require space-time Gaussians because we were not able to obtain reliable space-time Gaussians with available methods [16, 37].

3.2. Problem Formulation

Besides the two most recent velocities, we also extract the two most recent vertical coordinates $z_{t-1}^{(i)}$ and $z_t^{(i)}$ from each Gaussian (which are informative for modeling gravity and ground plane). We concatenate these into the per–point feature vector

$$\mathbf{f}_t^{(i)} = \left[\, \mathbf{v}_{t-1}^{(i)},\ \mathbf{v}_t^{(i)},\ z_{t-1}^{(i)},\ z_t^{(i)} \,\right]. \tag{4}$$

Given the set of per-point features $\{\mathbf{f}_t^{(i)}\}_{i=1}^N$, the network predicts the next frame Gaussian center $\{\mathbf{x}_{t+1}^{(i)}\}$ and rotation $\{\mathbf{R}_{t+1}^{(i)}\}$. The model can parameterize the center prediction either by predicting the next velocity or by predicting acceleration following [2]:

$$\hat{\mathbf{x}}_{t+1}^{(i)} = \mathbf{x}_t^{(i)} + \hat{\mathbf{v}}_{t+1}^{(i)}, \qquad \hat{\mathbf{x}}_{t+1}^{(i)} = \mathbf{x}_t^{(i)} + \mathbf{v}_t^{(i)} + \hat{\mathbf{a}}_t^{(i)}. \tag{5}$$

The rotation is updated via

$$\mathbf{R}_{t+1}^{(i)} = \Delta\mathbf{R}_t^{(i)}\, \mathbf{R}_t^{(i)}. \tag{6}$$

where $\Delta\mathbf{R}_t^{(i)} \in SO(3)$ is the predicted delta rotation.

3.3. Convolution-based Interaction Modeling

Prior work [12] modeled interactions among 3D points using PointConv [32, 33], which offers an efficient convolution operation over continuous 3D point sets. The 3D convolution operation is written as:

$$\mathbf{y}_0 = \mathbf{W}_l \mathrm{vec}\left(\frac{1}{|\mathcal{N}(\mathbf{p}_0)|}\sum_{\mathbf{p}_i \in \mathcal{N}(\mathbf{p}_0)} h(\mathbf{p}_i - \mathbf{p}_0)\mathbf{x}_i^\top\right) \quad (7)$$

where $\mathbf{p}_0 \in \mathbb{R}^3$ is a query point location for output features $\mathbf{y}_0 \in \mathbb{R}^{c_{\text{out}}}$. For each neighboring point $\mathbf{p}_i \in \mathbb{R}^3$ in the neighbor set $\mathcal{N}(\mathbf{p}_0)$, $\mathbf{x}_i \in \mathbb{R}^{c_{\text{in}}}$ denotes the input feature associated with that point, and $|\mathcal{N}(\mathbf{p}_0)|$ is the neighborhood size. $\mathbf{W}_l \in \mathbb{R}^{c_{\text{out}} \times (c_{\text{in}} c_{\text{mid}})}$ is a shared linear layer, $\mathrm{vec}(\cdot)$ converts a matrix into a vector, and $h(\cdot) \in \mathbb{R}^{c_{\text{mid}}}$ is an MLP that generates convolution kernel weights by embedding the relative position $(\mathbf{p}_i - \mathbf{p}_0)$.

The network in [12] interleaves Object PointConv and Relational PointConv layers. Object PointConv applies 3D continuous point convolution over points belonging to the same object, while Relational PointConv applies convolution between a point from one object and nearby points from other objects. Thus, in Eq. (7), the neighbor set $\mathcal{N}(\mathbf{p}_0)$ alternates between these two types depending on the layer, enabling the model to effectively learn force propagation effects both within objects and across objects.

3.4. Gaussian Integration

Unlike the perfect particle-level ground truth where each point is associated with a correct object ID in simulation environments, Gaussians produced by GS do not come with object IDs. However, we assume having multi-view 2D object segmentation masks that are consistent across camera views and over time, which are much more realistic thanks to foundational 2D segmentation models [20, 25] (data collection details in Sec. 4.2). We therefore extract object identity information of each Gaussian by measuring how much each Gaussian contributes to rendering each object mask.

In Gaussian Splatting, each Gaussian contributes to the color of multiple pixels. The rendering contribution of Gaussian i to pixel u in view c is written as:

$$\gamma_c^{(i)}(u) = \alpha_c^{(i)}(u) \prod_{j \in \mathcal{F}_c(i)} \big(1 - \alpha_c^{(j)}(u)\big), \quad (8)$$

where $\mathcal{F}_c(i)$ denotes the set of Gaussians that are in front of Gaussian i in view c.

For each object mask m in view c, we take the maximum rendering contribution of Gaussian i over all pixels u inside that mask:

$$\Gamma_{c,m}^{(i)} = \max_{u \in \text{mask}(c,m)} \gamma_c^{(i)}(u). \quad (9)$$

Given N Gaussians, C views, and M object masks per view, we obtain a tensor $\Gamma \in \mathbb{R}^{N \times C \times M}$ where each entry $\Gamma_{c,m}^{(i)}$ quantifies the strongest rendering evidence that Gaussian i belongs to object m in view c.

To assign an object ID to each Gaussian, we aggregate Γ across all camera views using a majority voting scheme. For each Gaussian i and each view c, we determine the most likely object ID for that view by

$$\text{ID}_c^{(i)} = \arg\max_{m \in \{1,\ldots,M\}} \Gamma_{c,m}^{(i)}. \quad (10)$$

We aggregate votes to determine the most frequently occurring ID across views:

$$\mathbf{v}^{(i)}[m] = \sum_{c=1}^{C} \mathbf{1}\Big\{\text{ID}_c^{(i)} = m\Big\} \quad (11)$$

where $\mathbf{v}^{(i)} \in \mathbb{R}^M$ is the vote count vector across views, and $\mathbf{1}\{\cdot\}$ denotes the indicator function, which is 1 if the condition is true and 0 otherwise. We convert the vote counts to a one–hot vector by

$$\mathbf{w}^{(i)} = \text{one-hot}\Big(\arg\max_m \mathbf{v}^{(i)}[m]\Big). \quad (12)$$

Alternatively, one can obtain a soft distribution instead of a one–hot vector. We provide ablations comparing different ID extraction approaches in the experiments section.

Unlike [12], where explicit object-specific neighborhoods were maintained to define nearest neighbors from the same object or different objects, we perform an object-agnostic KNN search first, and use the ID vectors to decide which neighbors contribute to each convolution. For Object PointConv, we define the affinity weight $m_{i,0} = (\mathbf{w}^{(i)})^\top \mathbf{w}^{(0)}$, and for Relational PointConv, we define $m_{i,0} = 1 - (\mathbf{w}^{(i)})^\top \mathbf{w}^{(0)}$. We then plug $m_{i,0}$ into Eq. (7) as:

$$\mathbf{y}_0 = \mathbf{W}_l \,\mathrm{vec}\left(\frac{1}{\sum_i m_{i,0}} \sum_{\mathbf{p}_i \in \mathcal{N}(\mathbf{p}_0)} m_{i,0}\, h(\mathbf{p}_i - \mathbf{p}_0)\, \mathbf{x}_i^\top\right). \quad (13)$$

This enables us to form the same object-level partitions that the convolution layers in Sec. 3.3 require (Object PointConv vs. Relational PointConv), without explicitly computing object-level neighborhoods, which would otherwise require inefficient per-object looping. This approach also naturally supports both discrete one-hot vectors and soft distributions for representing each Gaussian's object ID.

3.5. Network Architecture

We use a U-Net architecture that takes the original dense Gaussians as input and constructs hierarchical scene representations via two grid downsampling layers with grid sizes

of 2 cm and 5 cm, while interleaving Object and Relational PointConv layers throughout the hierarchy, similar to [12]. At the final resolution, a prediction head operating at the original dense Gaussian locations predicts the next velocity. The rotation update is predicted either by a follow-up rigid transformation fitting module [24] if it is enabled or directly by the prediction head using the 6D rotation representation [41]. Additional architectural details are provided in the supplementary material.

3.6. Training Details

We train our model using two types of supervision: rendering supervision and position regression. Neither source provides perfect or noise-free guidance. For rendering supervision, although we have clean image observations as ground truth, the extracted 3D Gaussians are imperfect, making it impossible to render future frames perfectly even with accurate motion predictions. For position supervision, the pseudo-labels obtained from long-term point tracking and video object segmentation also contain inherent noise. Therefore, we rely on both sources of supervision to train our model.

From the three input frames, we predict K future frames using our model. In this work, we set $K = 3$ for all experiments. During testing, the rollout can be extended for as long as needed. For $K > 1$, the prediction from the network is fed back into the model as input in an autoregressive manner. Given predictions for K frames of B scenes in a mini-batch, our loss function is written as:

$$\mathcal{L} = \frac{1}{BK} \sum_{i=1}^{B} \sum_{k=1}^{K} \left(\lambda_{\text{rend}} \, \mathcal{L}_{\text{render}}^{(i,k)} \; + \; \lambda_{\text{pos}} \, \mathcal{L}_{\text{pos}}^{(i,k)} \right). \quad (14)$$

Here, $\mathcal{L}_{\text{render}}^{(i,k)}$ is the rendering loss for the k-step prediction of the i-th example, and $\mathcal{L}_{\text{pos}}^{(i,k)}$ is the position regression loss for the k-step prediction of the i-th example. We use an L1 loss for $\mathcal{L}_{\text{render}}$ and the Huber loss for $\mathcal{L}_{\text{pos}}$. We set $\lambda_{\text{rend}} = 3$ and $\lambda_{\text{pos}} = 1$.

For training, we define one epoch as the iteration in which each scene is sampled 10 times when forming mini-batches. Frames are sampled either uniformly at random or according to a loss-based distribution which prioritizes higher-loss frames (i.e., hard example mining). All models reported in Sec. 5 are trained for 50 epochs. We use a batch size of $B = 12$ and train all models for 50 epochs with an initial learning rate of 0.001 using the Adam optimizer [13]. A step-based learning rate scheduler is applied, reducing the learning rate by a factor of 0.1 twice during training.

4. Real-World Collision Dataset

We present the first real-world dataset for learning multi-object collision dynamics, consisting of two distinct scenarios: falling cube stacks and bowling. In the falling cube stack scenario, towers built from toy cubes are subjected to a randomly applied external force, causing them to collapse. In the bowling scenario, a ball impacts up to ten toy bowling pins. Both scenarios are recorded on a tabletop setup using the toy sets shown in Fig. 2. With this, we hope to build a robust data collection pipeline that can easily scale up to more scenarios. We introduce more details about the dataset and how we label our data to enable model training.

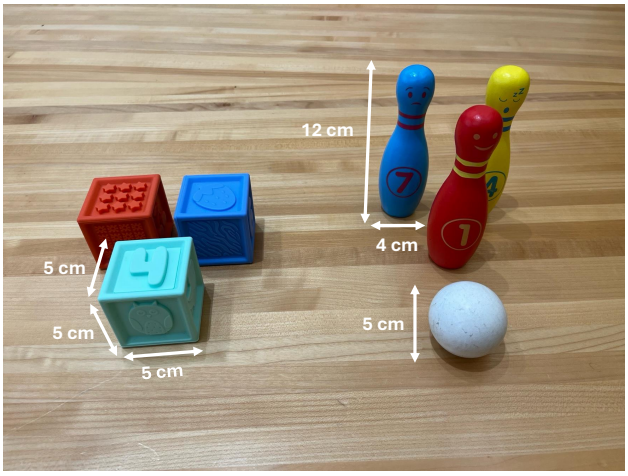

Figure 2. Objects used in our dataset. The falling-cube-stack scenario includes up to 10 cubes, while the bowling scenario includes one ball and up to 10 bowling pins. Overlaid annotations indicate real-world object dimensions.

4.1. Dataset Details

We collected 210 falling cube stack sequences and 292 bowling sequences. All sequences were captured in a calibrated multi-view setting using four Intel RealSense D455 cameras arranged around the table (Fig. 3). Camera intrinsics and extrinsics were estimated using a checkerboard placed on the table prior to recording. Images were recorded at a resolution of 640×480. Each scene consists of synchronized RGB-D images with known camera poses, as well as per-frame object segmentation masks whose object IDs are consistent across all camera views and over time. Additional details on data cleaning, annotation, and mask generation are provided in the next subsection.

4.2. Data Collection Pipeline

The overall data processing pipeline is shown in Fig. 4. After recording, each scene contains RGB-D image sequences from four calibrated cameras. For depth, instead of using the noisy depth maps produced by the RealSense cameras, we generate depth maps using FoundationStereo [28] from the RealSense left and right IR cameras. During capture, we disable the RealSense IR emitter, since the projected dot patterns may degrade FoundationStereo performance and limit its ability to generalize to our scenes.

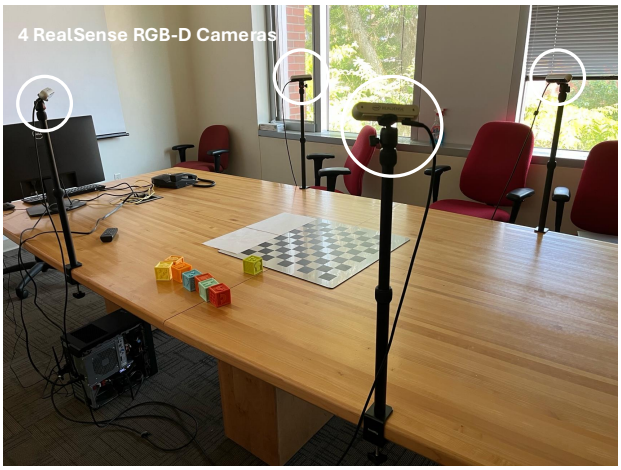

Figure 3. Data collection setup with four Intel D455 RealSense cameras. Camera intrinsics and extrinsics are estimated prior to recording using a checkerboard calibration.

For each recording, we manually annotate the first and last frame such that each sequence only contains the portion of the interaction after the initial impulse is applied by the human operator. In addition, we manually annotate a small set of 2D points (per object) at either the first or the last frame depending on visibility, and use these points as prompts for a SAM-style video segmentation and tracking method [20, 25]. Based on the annotation frame, masks are propagated either forward or backward through the sequence. To ensure consistent object IDs across all four camera views within a scene, we perform an additional cross-view data association step. Specifically, we select a reference view and find correspondences between object masks in the reference view and all other views via Hungarian matching, using the averaged 3D world coordinates of the 2D masks (e.g., 3D coordinates of pixels inside each 2D mask are obtained via depth and camera intrinsics/extrinsics and averaged). This produces multi-view object segmentation masks with consistent IDs semi-automatically, requiring only small amounts of manual annotations. While failures may occur (e.g., segmentation drift during tracking or inaccurate initial segmentation produced by point prompts), in practice we find such errors to be infrequent and acceptable as noise in the training data.

Our network takes 3D Gaussian trajectories as input. We initially experimented with 4D Gaussian Splatting methods designed for dynamic scenes [16, 31]. However, we found these methods to be slow, and the resulting 3D Gaussian trajectories to be unreliable. They often look reasonably only rendered in 2D, but the underlying 3D motion is inaccurate. Therefore, instead of extracting dynamic Gaussians directly, we estimate per-object 3D pose changes over time and apply those transforms to static 3D Gaussians to obtain the Gaussian trajectories.

We estimate per-object 3D poses using the iterative closest point (ICP) algorithm on partial point clouds in the world coordinate frame. ICP alone can fail when the initial alignment between two point clouds is poor. To obtain better initialization, we run an off-the-shelf 2D point tracking model [7] to produce point-level correspondences over time. Because each 2D pixel is associated with a 3D coordinate (via depth and camera calibration), we can lift tracked pixels into the 3D world frame, which gives us reliable initial correspondences. We initialize ICP with these correspondences and run ICP to convergence to recover the per-object pose trajectory. Once per-object poses are estimated for all frames, we propagate the static 3D Gaussians from any selected input frame through time to generate the 3D Gaussians trajectories used by our network. We generate the static 3D Gaussians in all frames by running Gaussian Splatting [11] on each scene, and store them as part of the dataset. This allows us to generate training examples from a random frame during training.

5. Experiment

In this section, we detail our experimental settings and report performance comparisons on our dataset, along with comprehensive ablation studies. We conduct experiments on the real-world dataset described in Sec. 4.2. For every video frame, we extract a set of 3D Gaussians, ensuring that the model always receives Gaussian-based inputs during both training and testing. Unless otherwise noted, all ablation baselines are trained under identical settings, and differences arise only from the variables we modify for each study. We run each method with three random seeds and report the mean and standard deviation.

5.1. Performance Metrics

Since we do not have ground-truth labels for evaluating the 3D reconstruction quality of each rollout, we primarily rely on rendering-based metrics such as PSNR, SSIM [27], and LPIPS [38], following prior work in 4D scene reconstruction [31, 37]. In addition, because we have pseudo ground-truth particle-motion labels generated from long-term point tracking, video segmentation, and stereo-depth estimation, we also evaluate rollouts using the position-accuracy metric δ_{avg}, which is commonly used in long-term point tracking [7, 16]. Due to the noise in the pseudo labels, we compute position accuracy at relatively high thresholds of 5 cm, 10 cm and 20 cm and report the average across the thresholds. Finally, we compute the Chamfer Distance (CD) (in cm) between the Gaussian positions produced by the rollout and those obtained during data preprocessing via GS. The CD is computed on the last frame of each sequence for which precomputed Gaussians are available, where these Gaussians serve as pseudo ground truth.

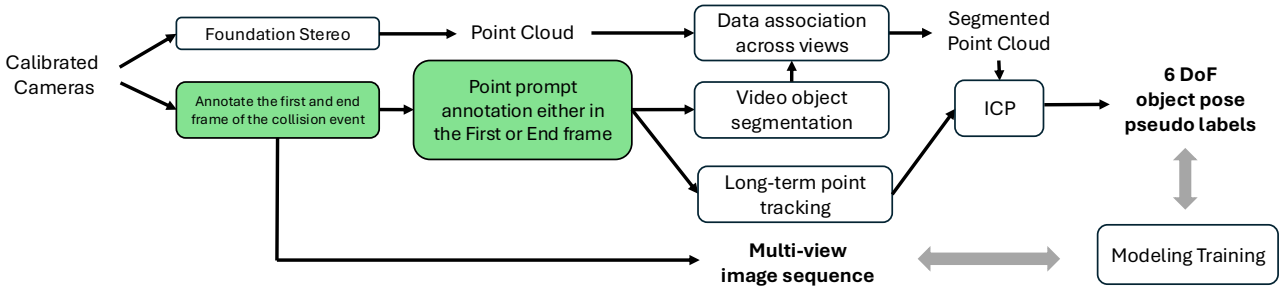

Figure 4. An overview of our data collection pipeline. It enables learning collision dynamics from real-world videos by providing two complementary learning signals for the model. Steps requiring manual annotation are highlighted in green.

5.2. Ablation Study

We investigate several design choices to better understand their impact on final performance and report the results in Table 1. From the overall results, we first note that rendering metrics are not very informative for evaluating neural dynamics models in our ablation setup, where the same input Gaussians are used across different methods. In contrast, CD and δ_{avg} are more discriminative, although these metrics are also imperfect since they rely on the pseudo ground truth. We discuss the limitations of current performance metrics in more detail in Sec. 6.

For the bowling scenario, the non-object-centric version of our network (i.e., without object–relation decomposition and without object-wise pose fitting) leads to unstable training and worse performance in terms of CD and δ_{avg}, as shown in the first row of the table. Similarly, removing object-wise pose fitting alone negatively impacts these metrics, as shown in the second row.

Next, since object IDs are assigned based on the rendering contribution of each Gaussian to segmentation masks across multiple views, we compare two alternatives: using a discrete object ID obtained via majority voting, and using a soft object-distribution representation (both supported by our model as described in Eq. (13)). This comparison corresponds to the third row and the second-to-last row of the table. Overall, the model using discrete IDs performs better than the one using soft IDs. This is likely because a single Gaussian typically represents only one moving object, except in rare boundary cases.

Regarding the loss design, using both position and rendering losses leads to more balanced performance across both the bowling and cube stacks scenarios, as shown in the second-to-last row. Since each scene contains multiple frames and only a subset is sampled per epoch, we further compare two frame selection strategies: uniform sampling and hard example mining. For hard example mining, after each epoch we evaluate the model on all frames and construct a sampling distribution proportional to per-frame loss, so that more difficult frames are sampled more frequently. As shown by comparing the second-to-last and last rows of Table 1, hard example mining improves performance in terms of CD and δ_{avg} but is not statistically significant.

5.3. Comparison with Published Work

Whitney et al. [29] is the closest work to ours, as it also learns multi-object collision dynamics using rendering supervision. However, since their code is not publicly available, we are unable to perform a direct quantitative comparison. Instead, we provide a qualitative assessment based on the videos available on their project webpage and our own rollout results. Their demonstrations on the Kubric MOVi-A/B/C datasets [5] include complex collision events involving multiple objects, where the model appears to struggle with reasoning about multi-object interactions. In contrast, our rollouts exhibit more physically plausible behavior in comparable multi-object scenarios, even though our model is trained on noisy real-world observations. Our rollout videos are available on our project webpage, which is linked in the abstract.

As discussed in Sec. 2, several prior works [15, 37] in robotics train GS-based dynamics models directly from real-world videos. However, these approaches primarily focus on modeling the effects of actions on a single manipulated object. For example, Lu et al. [15] model interactions using cross-attention between a robot-action token and scene tokens. If the action token is removed, the model no longer captures interactions, making it unsuitable for our multi-object, action-free setting. Similarly, the public implementation of [37] does not support multi-object interactions. Nevertheless, the underlying particle-dynamics module used in [37] is based on DPI [14], which has a publicly available multi-object implementation widely used in intuitive physics benchmarks within simulation environments [3, 23]. Therefore, we reimplement [37] using the DPI formulation and the Gaussian densification scheme proposed in [37], adapting it to our multi-object, action-free setting. We refer to this variant as GS-Dynamics*. More details on our adaptation of [14, 37] are provided in the supplementary material.

Table 1. Ablation results. The "Components" column includes the following abbreviations: Pos. Loss (position loss), Rend. Loss (rendering loss), O-centric (object-centric modeling), Disc. ID (discrete object ID), P. Fit (pose fitting), and HEM (hard example mining). "–" denotes components that are not applicable under a given configuration. Results are bolded if they are significantly better via an unpaired t-test.

Components						Bowling					Falling Cube Stacks				
Pos. Loss	Ren. Loss	O-Centric	Disc. ID	P. Fit	HEM	PSNR ↑	SSIM ↑	LPIPS ↓	CD ↓	δ_{avg} ↑	PSNR ↑	SSIM ↑	LPIPS ↓	CD ↓	δ_{avg} ↑
✓	✓	✗	–	✗	✗	**27.39** ± 0.08	0.975 ± 0.001	0.053 ± 0.004	17.06 ± 11.02	56.74 ± 8.12	**26.14** ± 0.08	0.973 ± 0.000	0.056 ± 0.000	10.76 ± 0.68	59.58 ± 0.93
✓	✓	✓	✓	✗	✗	**27.41** ± 0.10	0.976 ± 0.001	0.052 ± 0.003	11.15 ± 2.04	55.52 ± 5.12	**26.20** ± 0.03	0.974 ± 0.000	0.056 ± 0.000	10.82 ± 0.29	59.39 ± 0.63
✓	✓	✓	✗	✓	✗	27.19 ± 0.06	0.976 ± 0.000	0.050 ± 0.001	11.37 ± 2.09	61.64 ± 0.51	25.88 ± 0.04	0.971 ± 0.000	**0.055** ± 0.001	**10.37** ± 0.62	57.72 ± 1.14
✗	✓	✓	✓	✓	✗	**27.35** ± 0.79	**0.979** ± 0.001	0.056 ± 0.019	9.67 ± 0.05	**66.67** ± 0.13	**26.35** ± 0.22	**0.976** ± 0.000	**0.054** ± 0.001	10.40 ± 0.46	56.25 ± 1.05
✓	✗	✓	✓	✓	✗	**27.23** ± 0.06	0.976 ± 0.000	0.049 ± 0.000	9.85 ± 0.87	62.67 ± 0.72	25.85 ± 0.04	0.970 ± 0.000	**0.054** ± 0.001	**9.45** ± 0.12	**61.53** ± 0.60
✓	✓	✓	✓	✓	✗	**27.23** ± 0.14	0.976 ± 0.001	0.049 ± 0.001	9.74 ± 0.41	**63.54** ± 2.77	25.81 ± 0.02	0.971 ± 0.000	0.055 ± 0.000	9.98 ± 0.26	**60.82** ± 0.10
✓	✓	✓	✓	✓	✓	**27.31** ± 0.16	0.976 ± 0.001	0.047 ± 0.002	9.08 ± 0.53	**65.21** ± 2.15	25.92 ± 0.09	0.971 ± 0.000	**0.054** ± 0.000	**9.65** ± 0.27	**61.17** ± 0.89

Table 2. Performance Comparison on Bowling and Falling Cube Stacks with Published Baseline Reimplemented using our Pipeline and Data Preprocessing Approach

	Bowling					Falling Cube Stacks				
Method	PSNR ↑	SSIM ↑	LPIPS ↓	CD ↓	δ_{avg} ↑	PSNR ↑	SSIM ↑	LPIPS ↓	CD ↓	δ_{avg} ↑
GS-Dynamics*	27.34 ± 0.06	0.977 ± 0.000	0.047 ± 0.001	9.48 ± 0.13	62.94 ± 1.56	25.85 ± 0.06	0.971 ± 0.000	0.054 ± 0.001	10.07 ± 0.48	60.83 ± 0.82
Ours	27.31 ± 0.16	0.976 ± 0.001	0.047 ± 0.002	9.08 ± 0.53	65.21 ± 2.15	25.92 ± 0.09	0.971 ± 0.000	0.054 ± 0.000	9.65 ± 0.27	61.17 ± 0.89

We exclude approaches that use Gaussians generated by GS as input to an MPM-based simulation framework [34] from our comparisons. Although these methods can also produce rollouts from input Gaussians, they require per-object physical parameters (e.g., density, Young's modulus, friction coefficients), which must either be manually specified or estimated via system identification. The performance of MPM is highly sensitive to these parameters. As a result, a fair comparison would require jointly estimating these physical properties from observations using rendering supervision, which remains challenging and has only been explored in simpler settings than our collision scenarios in prior work.

Table 2 compares our approach with GS-Dynamics*. The results indicate that our proposed pipeline also enables training GS-Dynamics* to achieve competitive performance. Our model has some modest numerical advantage to it, while statistical significance is not established.

6. Limitations

A key limitation is the lack of reliable metrics for assessing physical correctness in Gaussian-based dynamics models. In our experiments, rendering-based metrics such as PSNR, SSIM, and LPIPS are not sufficiently discriminative. All methods operate on the same input Gaussians, resulting in visually similar renderings, and the foreground Gaussians occupy only a small portion of the image, further reducing sensitivity. Additionally, when predicted Gaussians do not align with ground-truth object pixels, rendering losses penalize such discrepancies uniformly, without considering whether the predicted configuration is physically plausible. Consequently, these metrics often fail to distinguish between physically meaningful and implausible dynamics.

Geometric metrics such as CD and δ_{avg} are more sensitive, but we observe that improvements in these metrics do not always correspond to qualitatively more realistic or physically plausible behavior in rollouts. Ideally, evaluation should reflect intuitive physical outcomes, such as whether objects remain stable, how many objects topple, or whether collisions are resolved correctly. However, defining such metrics is challenging in our setting because objects are represented as collections of Gaussians rather than explicit surfaces. As a result, extracting discrete object-level states (e.g., counting the number of cubes remaining on a table or pins that have fallen) is non-trivial and often unreliable.

Developing evaluation metrics that better capture physically meaningful behavior for Gaussian-based representations remains an open problem. We believe this is an important direction for future work, as improved metrics would enable more accurate assessment and comparison of learned dynamics models.

7. Conclusion

In this work, we presented the first framework for learning multi-object, action-free 3D collision dynamics from real-world videos using 3D Gaussian trajectories as an intermediate representation. Unlike existing approaches that rely on clean simulation labels, our method learns object interactions purely through rendering supervision and noisy position supervision derived from long-term point tracking and multi-view segmentation. To enable this, we introduced a real-world dataset consisting of around 500 multi-view videos capturing complex collision scenarios, including falling cube stacks and tabletop bowling. Looking forward, our approach opens up new possibilities for learning physically grounded 3D world models from raw and unstructured real-world video observations.

Acknowledgements

This work was partially supported by NSF grants 1751402 and 2321851, the DARPA TIAMAT grant HR0011-24-9-0423, and a 2023 Specialty Crop Block Grant from the Oregon Department of Agriculture.

References

[1] Kelsey R Allen, Tatiana Lopez-Guevara, Kimberly Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, Jessica Hamrick, and Tobias Pfaff. Physical design using differentiable learned simulators. *Neural Information Processing Systems (NeurIPS 2022)*, 2022. 1

[2] Kelsey R Allen, Tatiana Lopez Guevara, Yulia Rubanova, Kim Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, and Tobias Pfaff. Graph network simulators can learn discontinuous, rigid contact dynamics. In *CORL*, pages 1157–1167. PMLR, 2023. 3

[3] Daniel Bear, Elias Wang, Damian Mrowca, Felix Binder, Hsiao-Yu Tung, Pramod RT, Cameron Holdaway, Sirui Tao, Kevin Smith, Fan-Yun Sun, Fei-Fei Li, Nancy Kanwisher, Josh Tenenbaum, Dan Yamins, and Judith Fan. Physion: Evaluating physical prediction from vision in humans and machines. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran, 2021. 7

[4] Danny Driess, Ingmar Schubert, Pete Florence, Yunzhu Li, and Marc Toussaint. Reinforcement learning with neural radiance fields. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. 1, 2

[5] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: a scalable dataset generator. 2022. 7

[6] Jiaqi Han, Wenbing Huang, Hengbo Ma, Jiachen Li, Josh Tenenbaum, and Chuang Gan. Learning physical dynamics with subequivariant graph neural networks. In *NeuRIPS*, pages 26256–26268, 2022. 1

[7] Adam W. Harley, Yang You, Xinglong Sun, Yang Zheng, Nikhil Raghuraman, Yunqi Gu, Sheldon Liang, Wen-Hsuan Chu, Achal Dave, Pavel Tokmakov, Suya You, Rares Ambrus, Katerina Fragkiadaki, and Leonidas J. Guibas. AllTracker: Efficient dense point tracking at high resolution. 3, 6

[8] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International conference on robotics and automation (ICRA)*, 2019. 1

[9] Suning Huang, Qianzhong Chen, Xiaohan Zhang, Jiankai Sun, and Mac Schwager. Particleformer: A 3d point cloud world model for multi-object, multi-material robotic manipulation. In *9th Annual Conference on Robot Learning*, 2025. 1

[10] Hanxiao Jiang, Hao-Yu Hsu, Kaifeng Zhang, Hsin-Ni Yu, Shenlong Wang, and Yunzhu Li. Phystwin: Physics-informed reconstruction and simulation of deformable objects from videos. *ICCV*, 2025. 3

[11] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 1, 2, 3, 6

[12] Chanho Kim and Li Fuxin. Object dynamics modeling with hierarchical point cloud-based representations. In *CVPR*, 2024. 1, 3, 4, 5

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 5

[14] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019. 1, 7

[15] Guanxing Lu, Baoxiong Jia, Puhao Li, Yixin Chen, Ziwei Wang, Yansong Tang, and Siyuan Huang. Gwm: Towards scalable gaussian world models for robotic manipulation. *Proceedings of International Conference on Computer Vision (ICCV)*, 2025. 1, 3, 7

[16] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. 3, 6

[17] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2

[18] NVIDIA, :, Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, Daniel Dworakowski, Jiaojiao Fan, Michele Fenzi, Francesco Ferroni, Sanja Fidler, Dieter Fox, Songwei Ge, Yunhao Ge, Jinwei Gu, Siddharth Gururani, Ethan He, Jiahui Huang, Jacob Huffman, Pooya Jannaty, Jingyi Jin, Seung Wook Kim, Gergely Klár, Grace Lam, Shiyi Lan, Laura Leal-Taixe, Anqi Li, Zhaoshuo Li, Chen-Hsuan Lin, Tsung-Yi Lin, Huan Ling, Ming-Yu Liu, Xian Liu, Alice Luo, Qianli Ma, Hanzi Mao, Kaichun Mo, Arsalan Mousavian, Seungjun Nah, Sriharsha Niverty, David Page, Despoina Paschalidou, Zeeshan Patel, Lindsey Pavao, Morteza Ramezanali, Fitsum Reda, Xiaowei Ren, Vasanth Rao Naik Sabavat, Ed Schmerling, Stella Shi, Bartosz Stefaniak, Shitao Tang, Lyne Tchapmi, Przemek Tredak, Wei-Cheng Tseng, Jibin Varghese, Hao Wang, Haoxiang Wang, Heng Wang, Ting-Chun Wang, Fangyin Wei, Xinyue Wei, Jay Zhangjie Wu, Jiashu Xu, Wei Yang, Lin Yen-Chen, Xiaohui Zeng, Yu Zeng, Jing Zhang, Qinsheng Zhang, Yuxuan Zhang, Qingqing Zhao, and Artur Zolkowski. Cosmos world foundation model platform for physical ai, 2025. 1

[19] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez,

and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *ICLR*, 2021. 1

[20] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. 2, 4, 6

[21] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *ICML*, 2020. 1

[22] Haochen Shi, Huazhe Xu, Samuel Clarke, Yunzhu Li, and Jiajun Wu. Robocook: Long-horizon elasto-plastic object manipulation with diverse tools. *arXiv preprint arXiv:2306.14447*, 2023. 1

[23] Hsiao-Yu Tung, Mingyu Ding, Zhenfang Chen, Daniel M. Bear, Chuang Gan, Joshua B. Tenenbaum, Daniel L. K. Yamins, Judith Fan, and Kevin A. Smith. Physion++: evaluating physical scene understanding that requires online inference of different physical properties. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2023. Curran Associates Inc. 7

[24] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, 1991. 5

[25] Jovana Videnovic, Alan Lukezic, and Matej Kristan. A distractor-aware memory for visual object tracking with SAM2. In *CVPR*, 2025. 2, 4, 6

[26] Jiaxu Wang, Jingkai Sun, Junhao He, Ziyi Zhang, Qiang Zhang, Mingyuan Sun, and Renjing Xu. Del: Discrete element learner for learning 3d particle dynamics with neural rendering. In *Advances in Neural Information Processing Systems*, pages 45703–45736. Curran Associates, Inc., 2024. 3

[27] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004. 6

[28] Bowen Wen, Matthew Trepte, Joseph Aribido, Jan Kautz, Orazio Gallo, and Stan Birchfield. Foundationstereo: Zero-shot stereo matching. *CVPR*, 2025. 5

[29] William F Whitney, Tatiana Lopez-Guevara, Tobias Pfaff, Yulia Rubanova, Thomas Kipf, Kim Stachenfeld, and Kelsey R Allen. Learning 3d particle-based simulators from RGB-d videos. In *The Twelfth International Conference on Learning Representations*, 2024. 3, 7

[30] William F Whitney, Jake Varley, Deepali Jain, Krzysztof Marcin Choromanski, Sumeet Singh, and Vikas Sindhwani. Modeling the real world with high-density visual particle dynamics. In *8th Annual Conference on Robot Learning*, 2024. 2

[31] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20310–20320, 2024. 6

[32] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, pages 9621–9630, 2019. 3, 4

[33] Wenxuan Wu, Li Fuxin, and Qi Shan. Pointconvformer: Revenge of the point-based convolution. In *CVPR*, pages 21802–21813, 2023. 3, 4

[34] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4389–4398, 2024. 1, 3, 8

[35] Haotian Xue, Antonio Torralba, Joshua B. Tenenbaum, Daniel LK Yamins, Yunzhu Li, and Hsiao-Yu Tung. 3d-intphys: Towards more generalized 3d-grounded visual intuitive physics under challenging scenes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 2

[36] Kaifeng Zhang, Baoyu Li, Kris Hauser, and Yunzhu Li. Particle-grid neural dynamics for learning deformable object models from rgb-d videos. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025. 1, 3

[37] Mingtong Zhang, Kaifeng Zhang, and Yunzhu Li. Dynamic 3d gaussian tracking for graph-based neural dynamics modeling. In *8th Annual Conference on Robot Learning*, 2024. 1, 3, 6, 7

[38] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

[39] Mikel Zhobro, Andreas René Geist, and Georg Martius. Learning 3d-gaussian simulators from rgb videos, 2025. 3

[40] Licheng Zhong, Hong-Xing Yu, Jiajun Wu, and Yunzhu Li. Reconstruction and simulation of elastic objects with spring-mass 3d gaussians. *European Conference on Computer Vision (ECCV)*, 2024. 3

[41] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 5