

A 3DGSIM ADDITIONAL DETAILS

A.1 UNPROJECTING PIXEL-ALIGNED FEATURES VIA FiLM CONDITIONING

To transform the view-dependent pixel-aligned features \hat{f}'_i into a consistent 3D representation, we use a multilayer perceptron (MLP) with Feature-wise Linear Modulation (FiLM). FiLM conditioning enables the MLP to adapt its processing of \hat{f}'_i based on geometric context, such as camera viewpoint and depth.

Specifically, FiLM computes a scale and bias using a conditioning network γ that takes as input a geometric context vector \mathbf{x}_i —which includes depth, density, and pixel shift—as well as the Plücker ray encoding $\mathbf{r}_i = [\mathbf{o}_i \times \mathbf{d}_i \mid \mathbf{d}_i]$, where \mathbf{o}_i and \mathbf{d}_i denote the origin and direction of the viewing ray:

$$\text{scale}_i, \text{bias}_i = \gamma(\mathbf{x}_i, \mathbf{r}_i). \quad (\text{S1})$$

These parameters modulate the activations of the MLP processing \hat{f}'_i through FiLM layers:

$$f_i = \text{MLP}(\hat{f}'_i; \text{scale}_i, \text{bias}_i), \quad \text{FiLM}(h) = \text{scale}_i \odot h + \text{bias}_i, \quad (\text{S2})$$

where h denotes a hidden activation and \odot is element-wise multiplication.

This setup allows the network to unproject image-aligned features into canonical 3D space while respecting scene geometry and view direction.

A.2 PTV3: SCALABLE POINT CLOUD TRANSFORMATIONS

Point cloud serialization At the core of PTV3 lies “point cloud serialization”, an algorithm that transforms an unstructured point cloud into ordered points. This process begins by discretizing 3D space into a uniform grid of points. As illustrated in Fig. 5, these point are then connected using a space-filling curve – a path that traverses each grid point exactly once while preserving spatial proximity as much as possible. Each point p_i is assigned an integer code s_i , representing its position within a space-filling curve, via the mapping

$$s_i = \phi^{-1}(\lfloor p_i/G \rfloor) \quad (\text{S3})$$

with $\phi^{-1} : \mathcal{Z}^N \mapsto \mathcal{Z}$ and grid size $G \in \mathbb{R}$. The points in the clouds are then ordered by their respective code s_i , yielding a serialized point cloud (SPC)

$$S_i(t_k) = \{(s_i, \tilde{g}_i)\}_{i=1}^{N_k}. \quad (\text{S4})$$

While this approach may not preserve local connectivity as precisely as kNN groupings, [Wu et al. \(2024b\)](#) emphasizes that the slight loss in spatial precision is outweighed by a significant gain in computational efficiency. To obtain diverse spatial connections between points, PTV3 shuffles between four different space filling curve patterns to obtain SPCs from which patches are computed and varies the patch computation through integer dilation.

Patch grouping PTV3 partitions the SPC $S_i(t_k)$ into equally sized patches and applies self-attention within each patch. To ensure divisibility, patches that do not align with the specified size are padded by borrowing points from neighboring patches.

Conditional embeddings and patch attention Besides the computational efficiency of SPC over kNN, its main advantage lies in the compatibility with standard dot-product attention mechanisms. To understand this, we first examine how the standard PTV3 architecture computes particle-wise predictions from a single point cloud at a single time step. The process begins by extracting particle-wise embeddings E_i for each serialized point (s_i, \tilde{g}_i) using a sparse convolutional neural network (CNN). Next, the embedded SPCs are progressively down sampled via grid pooling before being grouped and shuffled into patch pairs. Conditional positional embeddings (xCPE) are then added to the embeddings, followed by layer normalization and a patch-wise attention layer predicting the change in the embeddings ΔE_i . The pooling, patch shuffling, and attention blocks are arranged in a U-net [Ronneberger et al. \(2015\)](#) like architecture that first reduces the size of the SPCs in an encoding step and then mirrors this architecture. In 3DGSim, the final layer of the dynamics model predicts the change in the particle positions Δp_i and the change in their features Δf_i .

A.3 ARCHITECTURES

This section describes additional implementation details that were not explicitly mentioned in the main paper.

Feature Encoding Network In 4.2 we describe the feature encoding network that transforms pixel-aligned features into view-independent the latent features. To regress the FiLM conditioning we use a 2-layer CNN with GELU activation, kernel of size 3 and channel dimensions (10, 20). The first dimension also matches the size of the conditioning vector.

Particle Wise MLP At the end of the dynamics model, the embedding of each particle is mapped back to the particle latent features. For that we use an MLP with shapes (128, 128) and GELU activation between each layer.

B ABLATIONS

For the ablation studies of 3DGSim, only the elastic dataset is used. The default configuration uses the latent representation, 4-step past conditioning, 12-step future rollouts, 4 input views, 5 target views, and a total of 12 cameras for training, unless otherwise specified. Any deviations from these parameters are explicitly stated or made clear within the context of the respective ablation.

This analysis provides detailed insights into design and strategic choices that inform future improvements of our approach.

B.1 ROLLOUT LENGTH.

| Rollout Length | PSNR Future | PSNR Past |
|-----------------|------------------------------------|------------------------------------|
| 2 steps | 26.43 ± 3.48 | 35.70 ± 2.51 |
| 4 steps | 28.83 ± 4.96 | 35.25 ± 2.53 |
| 8 steps | 30.64 ± 3.23 | 33.86 ± 2.17 |
| 12 steps | 33.15 ± 3.51 | 34.55 ± 2.26 |

Table S1: Rollout Length

We evaluate the influence of prediction rollout length during training (2, 4, 8, and 12 steps). Consistent with expectations, results improve significantly as the rollout length increases, reaching a peak performance at 12 steps with a PSNR Future of 33.15 ± 3.51 . Extending the number of rollout steps enhances the model’s predictive capability but leads to significant memory requirements. Employing regularization methods such as random-walk noise injection or diffusion techniques or even other modality (see later) can help reduce the required rollout steps.

B.2 CAMERA SETUP.

| Setup | PSNR Future | PSNR Past |
|---------------------------------|------------------------------------|------------------------------------|
| Explicit 3 views out of 6 | 21.02 ± 1.78 | 16.86 ± 0.83 |
| Latent 3 views out of 6 | 31.60 ± 3.09 | 32.55 ± 2.12 |
| Latent 4 views out of 12 | 33.15 ± 3.51 | 34.55 ± 2.26 |

Table S2: Camera Setup (85k steps)

To approximate a realistic scenario suitable for real-world deployment, we investigate performance with reduced camera setups. Interestingly, the latent representation models achieve robust performance even when trained with 3 views out of 6 total cameras (PSNR 33.15), whereas explicit representation models degrade significantly (PSNR drops to 21.02) due to convergence of the encoder to poor local minima. This local minimum manifests as camera-specific overfitting, where the

model erroneously predicts particle arrangements forming planar, screen-aligned shapes. While this artificially reduces the training loss of target viewpoints, it undermines the true representation quality and disrupts convergence to a consistent 3D reconstruction. Further restricting the setup to only 2 views out of 4 cameras results in unsuccessful training for both latent and explicit models, indicating that very limited camera setups demand careful placement or preliminary encoder pre-training, which should be investigated in a future work.

B.3 SEGMENTATION MASKS.

| Segmentation | PSNR Future | PSNR Past |
|---------------|------------------|------------------|
| Without masks | 32.66 ± 3.43 | 39.08 ± 3.18 |
| With masks | 33.15 ± 3.51 | 34.55 ± 2.26 |

Table S3: Segmentation Masks

While our final models use segmentation masks for static objects (e.g., ground surfaces), we explore training without these masks to test model reliance on explicit segmentation. We find only a slight reduction in performance (32.66 vs. 33.15), demonstrating the models' capability to implicitly infer static scene regions directly from raw RGB inputs. Thus, explicit segmentation masks are helpful but not strictly essential.

B.4 MODALITY CONFIGURATIONS

| Modality | PSNR Future | PSNR Past |
|----------|------------------|------------------|
| 4-1-2-6 | 32.59 ± 3.22 | 33.39 ± 2.21 |
| 4-4-1-3 | 31.98 ± 3.78 | 34.03 ± 2.39 |
| 4-1-1-12 | 33.15 ± 3.51 | 34.55 ± 2.26 |

Table S4: Input Modality

Different input modality configurations were tested. Here, we adapt the notation "a-b-c-d", each varying the temporal span of input conditioning a is the number of particle frames representing the state, b indicates the number of backward frames used to predict c future steps, and total rollout steps during training d , leading to $b \cdot c \cdot d$ rollout steps per training step. Both variants ("4-1-2-6", "4-4-1-3" attain competitive performance (PSNR of 32.59 and 32.27 respectively), significantly reducing the computational load compared to longer standard rollouts. This highlights promising avenues for future investigation, emphasizing balance between computational efficiency and performance quality.

B.5 GRID RESOLUTION

| Grid Size | PSNR Future | PSNR Past |
|--------------|------------------------------------|------------------------------------|
| 0.002 | 25.39 ± 2.95 | 32.28 ± 2.68 |
| 0.004 | 33.15 ± 3.51 | 34.55 ± 2.26 |
| 0.008 | 25.40 ± 3.84 | 30.78 ± 2.63 |
| 0.0012 | 24.06 ± 3.53 | 31.74 ± 2.56 |

Table S5: Grid Resolution.

We test a series of grid resolutions (0.002, 0.004, and 0.008), observing optimal results at 0.004 with a PSNR of 33.15. Both higher (0.008) and finer resolutions (0.002) degrade performance, suggesting an optimal balance achieved at 0.004 between detail preservation and computational complexity for our scene size.

B.6 TEMPORAL MERGER.

| Temporal Merger Setup | PSNR Future | PSNR Past |
|---|------------------------------------|------------------------------------|
| [1,1,2,2,..] with embedding | 27.55 ± 3.22 | 31.68 ± 2.60 |
| [1,1,4,..] with embedding | 26.79 ± 2.94 | 32.21 ± 2.73 |
| [1,2,2,..] without embedding | 25.07 ± 3.22 | 31.16 ± 2.59 |
| [1,2,2,..] with embedding (120k) | 33.15 ± 3.51 | 34.55 ± 2.26 |
| [1,1,1,2,2] with embedding | 18.87 ± 1.50 | 18.09 ± 1.45 |
| [1,4,..] with embedding | 18.19 ± 1.29 | 18.33 ± 1.48 |

Table S6: Temporal Merger. The models are trained for 80k steps if not specified otherwise.

We experiment with various temporal merging strides for each encoder stage combined with embedding options of timestep position encoding. Since we only train with 4 past steps, the strides ".." don't influence the results. After 80k iterations, results clearly indicate two critical factors for success: the use of learned positional embeddings and timing of merging operations. Optimal results occur when merging temporal information only after early spatial processing stages ([1,2,2,..]), whereas early or too-late merging drastically diminishes performance. Poor outcomes with late merging likely arise due to spatial pooling operations that dilute vital temporal distinctions before merging.

C POSITIONING 3DGSIM AMONG EXISTING APPROACHES

To clarify the scope of our contributions, we distinguish 3DGSim from methods focused on dynamic scene reconstruction, and then contrast it with approaches that augment reconstructed 3D scenes with simulation capabilities.

C.1 CLARIFICATION ON THE DISTINCTION BETWEEN 3DGSIM AND DYNAMIC SCENE RECONSTRUCTION METHODS

Here, we elucidate the fundamental differences between our approach, 3DGSim, and dynamic scene reconstruction methods such as 4DGS [Wu et al. \(2024a\)](#) and DeformableGS [Bae et al. \(2024\)](#).

Temporal Characteristics: Dynamic scene reconstruction techniques, including 4DGS and DeformableGS, aim to reconstruct a temporally-varying 3D representation from a complete set of video frames, encompassing past, present, and future data. These methods are inherently temporally *non-causal*, as they leverage the entire video sequence to infer and optimize a 4D (3D + time) representation of the observed events. Their primary function is to interpolate or reconstruct what has already occurred within the video.

Predictive vs. Reconstructive Nature: In stark contrast, 3DGSim is designed for temporally *causal prediction*. Given a set of scene representations from a few past timesteps, our model's objective is to predict the future evolution of the scene. This prediction is made without any access to future video frames. This predictive capability necessitates that the model develops an understanding of the underlying physics governing the scene's dynamics.

C.2 COMPARISON TO PHYSGAUSSIAN AND PHYSDREAMER

PhysGaussian [Xie et al. \(2023\)](#) and PhysDreamer [Zhang et al. \(2024b\)](#) reconstruct 3D representations from multi-view images of static scenes and then simulate mesh-node dynamics via the Material Point Method (MPM). While these methods are effective for VR and per-scene content creation, a direct comparison to 3DGSim is not feasible for several reasons:

PhysGaussian does not learn dynamics. PhysGaussian reconstructs a 3D representation from a *static* scene and simulates particles with MPM. In our experiments, training images are generated using a combination of MPM simulators. One could attempt to tune the PhysGaussian MPM simulator to match the data-generating simulation, but hand-tuning MPM parameters to match the motion of another simulation is notoriously difficult. In contrast, 3DGSim learns to simulate object dynamics

directly from video without requiring physical priors. *While our method can be extended to learn different material modalities and collisions, it is unclear how to tackle this problem with MPM.*

PhysDreamer performs per-scene optimization. Our method is a feed-forward world model trained once and reused across scenes. By contrast, PhysDreamer requires training a separate model for each trajectory.

PhysDreamer does not simulate collisions. As the authors note: “In this work, we restrict our scope to elastic objects without collisions.” In our experiments, 3DGSim accurately simulates both elastic and rigid collisions.

In PhysDreamer (and MPM in general), boundary conditions must be defined by hand. Static parts of objects are manually set to be static. 3DGSim learns constrained dynamics directly from videos.

MPM requires very small simulation steps, making full-trajectory backpropagation impractical. For instance, in PhysDreamer the timestep is $\Delta t = 1 \times 10^{-4}$ with 768 sub-steps per frame. Training is split into two stages: 1) optimizing initial velocities on a few frames, 2) estimating material parameters with fixed velocities. In the second stage, gradients are truncated to flow only one frame backward to avoid explosion/vanishing gradients. As the authors state [Zhang et al. \(2024b\)](#):

“Rather than optimizing the material parameters and initial velocity jointly, we split the optimization into two stages for better stability and faster convergence. In particular, in the first stage, we randomly initialize the Young’s modulus for each Gaussian particle and freeze it. We optimize the initial velocity of each particle using only the first three frames of the reference video. In the second stage, we freeze the initial velocity and optimize the spatially varying Young’s modulus. During the second stage, the gradient signal only flows to the previous frame to prevent gradient explosion/vanishing.”

In contrast, 3DGSim jointly learns 3D reconstruction and dynamics simulation in a *fully end-to-end* manner, with *backpropagation through entire trajectories*, without requiring staged training or gradient truncation. Its transformer-based architecture enables significantly larger timesteps. Empirically, 3DGSim achieves accurate long-range predictions while being *1–2 orders of magnitude faster to train and simulate*.

C.3 SUMMARY:

The objective of methods like 4DGS and DeformableGS is scene reconstruction. They are not designed to forecast how a physical scene will evolve. Conversely, 3DGSim is a particle-based simulator that learns physics solely from video, enabling forecasting. Approaches like PhysGaussian and PhysDreamer reconstruct 3D representations from static scenes and then use material-point method (MPM) for simulation, they do not incorporate learning of the dynamics in the same manner. In fact, the authors of PhysGaussian emphasize that their framework is unable to handle collisions, a key aspect of physical simulation.

Our work instead tackles vision-based physics learning, which poses a distinct challenge beyond dynamic 3D scene reconstruction. Because of this difference, no suitable baselines currently exist, and we therefore benchmark against video generation models.

D VISUALIZATIONS

In this section we show rolled out dynamic prediction for different scenarios. If not otherwise stated, the first row in each image shows the ground truth predictions. First we depict scene editability and composability followed by several generalizations to multi body and visualization for each dataset. Finally we depict several rollouts from the test set for every type of dynamics for both 3DGSim and CosmosFT. CosmosFT is conditioned on past 5 frames and following prompts:

| Prompt |
|--|
| <i>A rigid body falling on a circular gray ground</i> |
| <i>A soft body falling on a circular-gray-ground</i> |
| <i>A rigid body falling on a red rectangular cloth which is fixed on its corners</i> |

Table S7: Prompts used to condition Cosmos and CosmosFT.

Videos of the predictions below are available in the supplementary material.

D.1 SCENE EDITABILITY

D.1.1 3DGSIM



Figure S1: Ground Removal

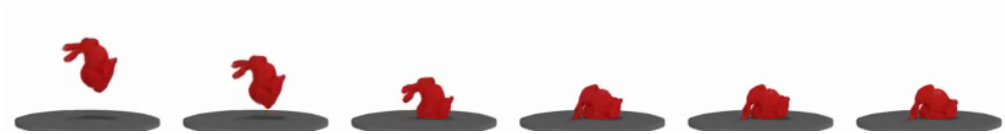


Figure S2: Higher Throw

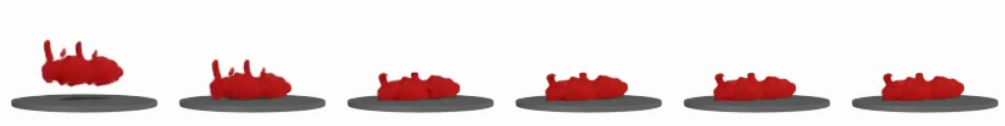


Figure S3: Parallel Simulation

D.1.2 COSMOSFT



Figure S4: CosmosFT: When ground is removed, objects often remain suspended.

D.2 GENERALIZATION TO MULTIPLE BODIES

D.2.1 3DGSIM

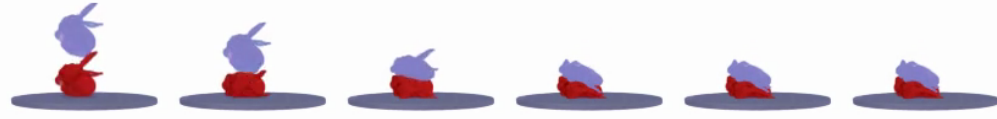


Figure S5: Scene with 2 bunnies.

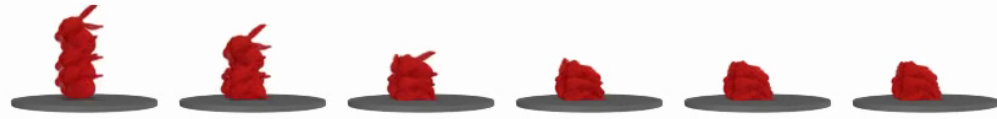


Figure S6: Scene with 3 bunnies.

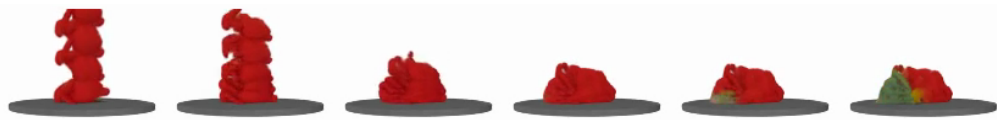


Figure S7: Scene with 5 bunnies. Interestingly, the combined weight of five objects slowly pushes the particles through the table causing color changes.



Figure S8: Scene with 5 worms.

D.2.2 COSMOSFT



Figure S9: Cosmos FT: multiple worms are morphed into one single object before colliding with the ground.

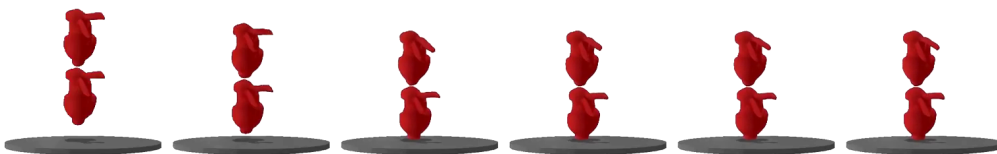


Figure S10: CosmosFT: two bunnies levitate above each just before colliding with the ground.

D.3 VISUALIZATION OF CLOTH SIMULATIONS

Here we present various cloth simulations for different objects. These are all trajectories from the test-set. The first row is the ground-truth and the second row the prediction.

D.3.1 3DGSIM



Figure S11: Dragon

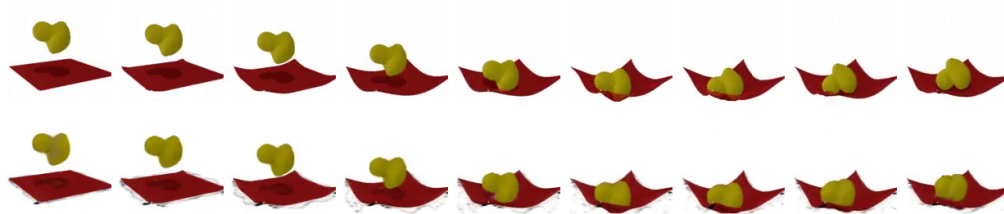


Figure S12: Duck



Figure S13: Cow

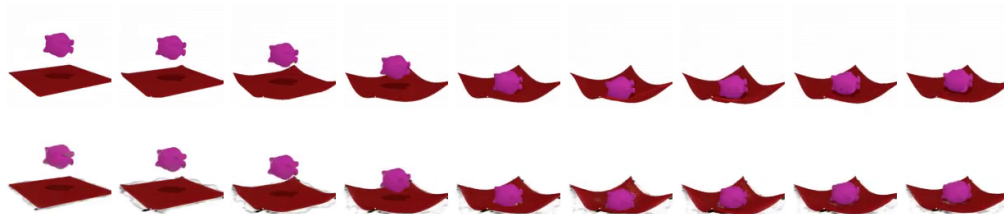


Figure S14: Devil

D.3.2 COSMOSFT

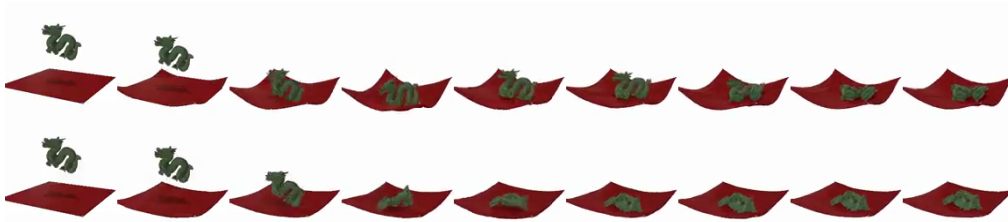


Figure S15: Dragon

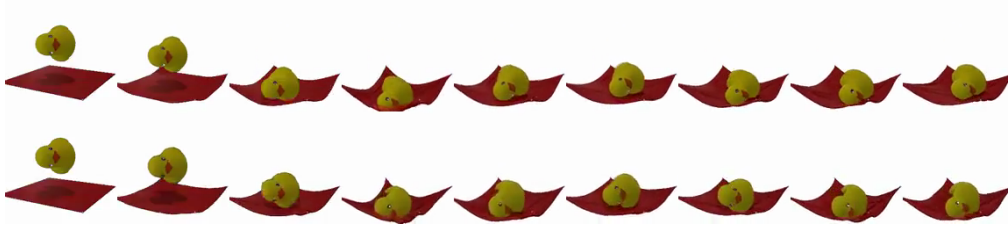


Figure S16: Duck

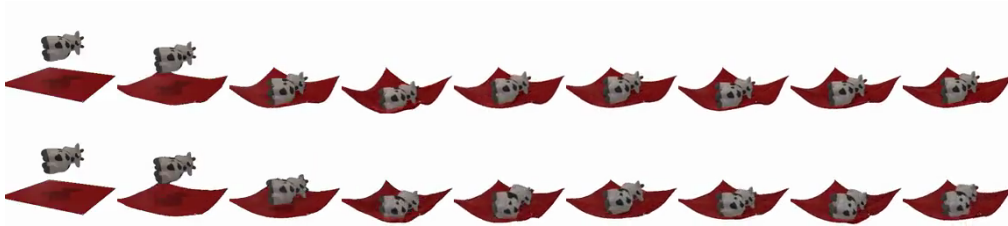


Figure S17: Cow

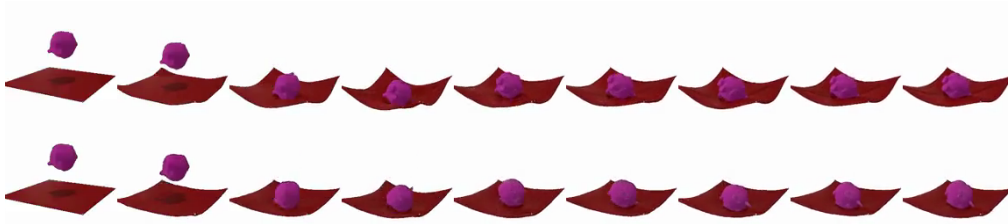


Figure S18: Devil

D.4 VISUALIZATION FOR ELASTIC DYNAMICS

Here we present various elastic simulations for different objects. These are all trajectories from the test-set. The first row is the ground-truth and the second row the prediction.

D.4.1 3DGSIM



Figure S19: Cow

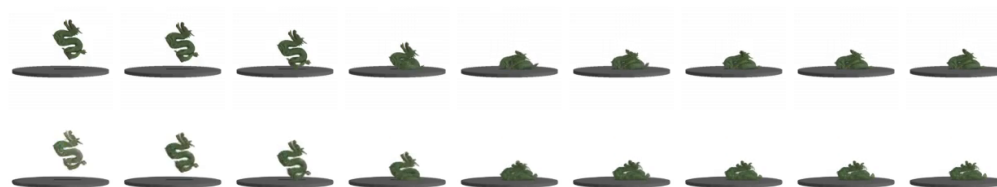


Figure S20: Dragon

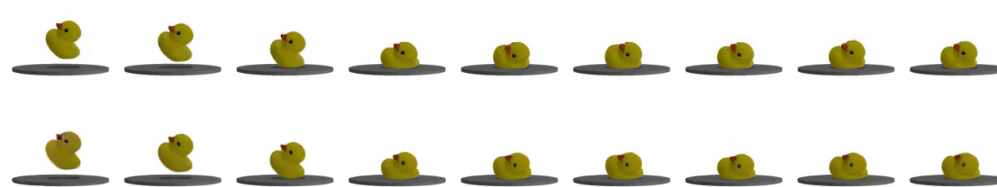


Figure S21: Duck



Figure S22: Pig



Figure S23: Worm

D.4.2 COSMOSFT



Figure S24: Cow

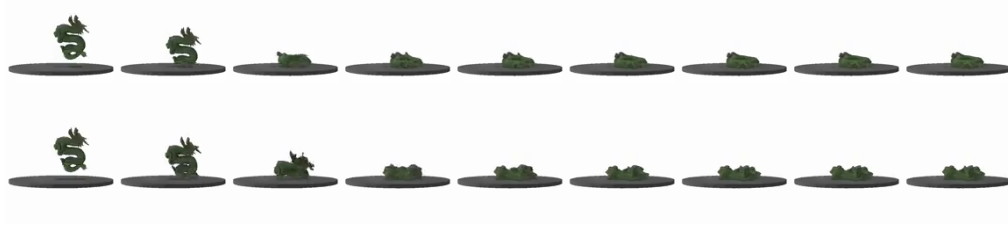


Figure S25: Dragon

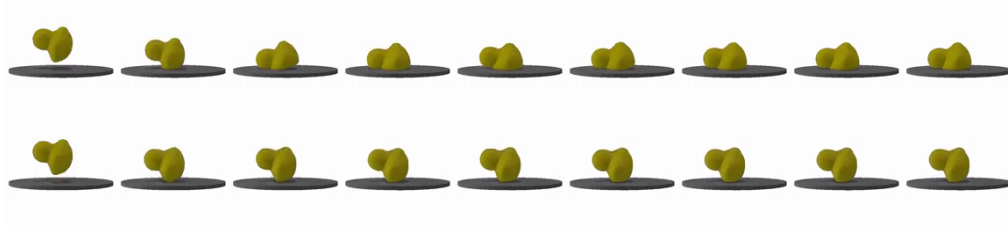


Figure S26: Duck



Figure S27: Pig

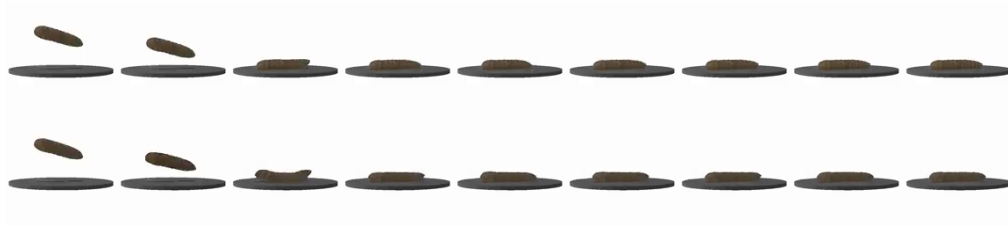


Figure S28: Worm

D.5 VISUALIZATION FOR RIGID BODY DYNAMICS

Here we present various rigid simulations for different objects. These are all trajectories from the test-set. The first row is the ground-truth and the second row the prediction.

D.5.1 3DGSIM

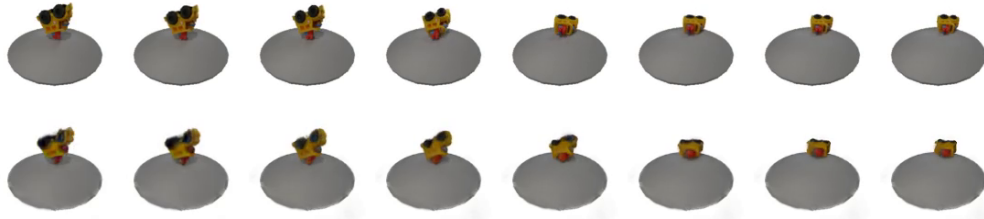


Figure S29: Car

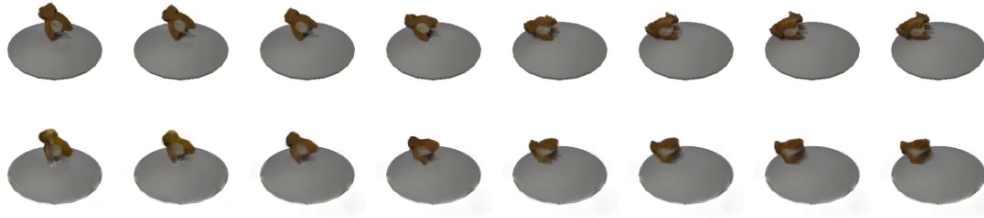


Figure S30: Squirrel



Figure S31: Shoe

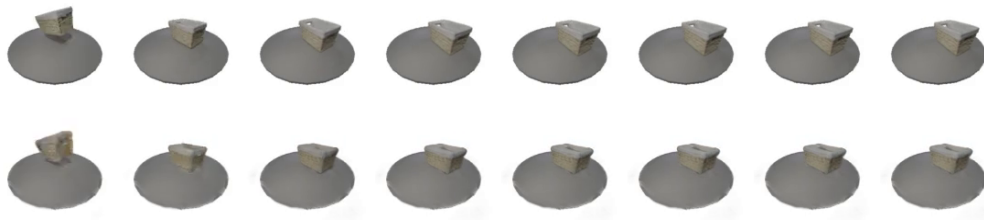


Figure S32: Box

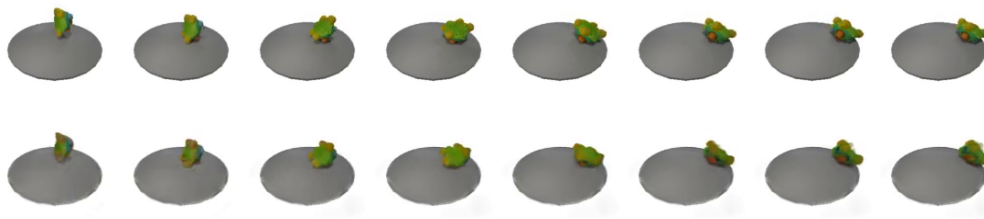


Figure S33: Turtle

D.5.2 COSMOSFT

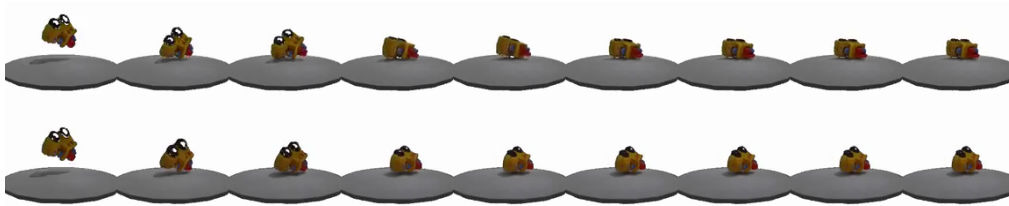


Figure S34: Car

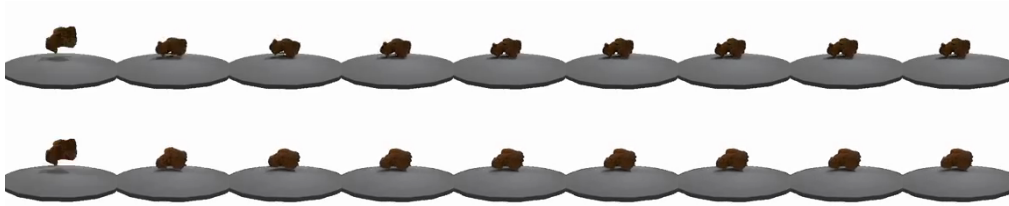


Figure S35: Squirrel

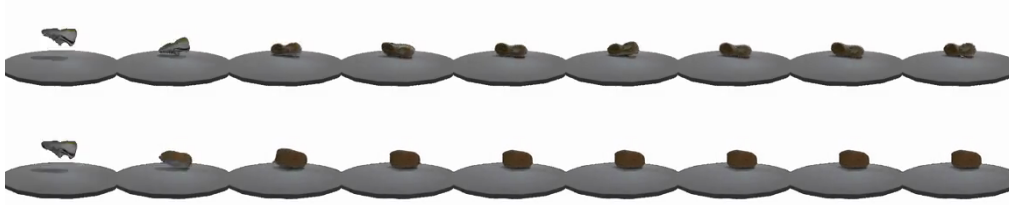


Figure S36: Shoe

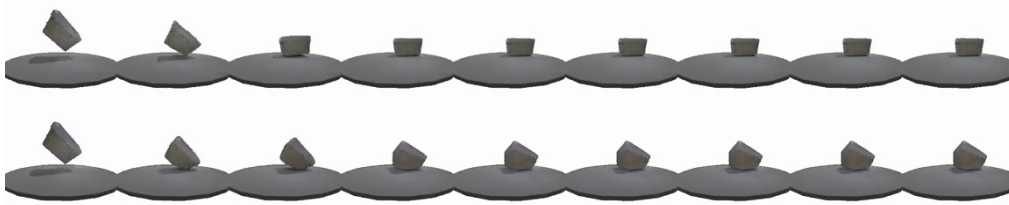


Figure S37: Box

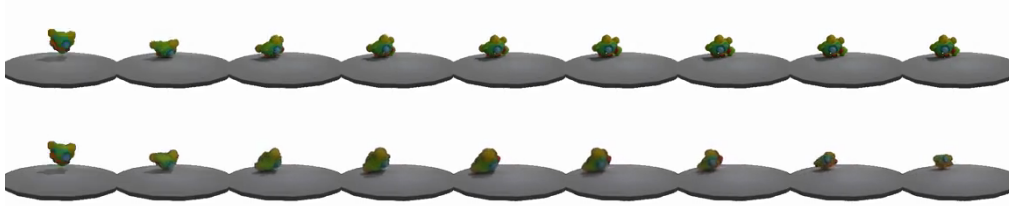


Figure S38: Turtle