Learning a Particle Dynamics Model with Real-world Video

Supplementary Material

1. Network Architecture Details

We adopt a U-Net architecture following [2]. Each stage of the U-Net contains an interaction block composed of an object PointConv layer followed by a relational PointConv layer. The object PointConv layer may change the spatial resolution depending on the stage via grid downsampling or upsampling, whereas the relational PointConv layer always preserves the input resolution. During downsampling, the feature dimensionality is doubled, and during upsampling it is halved. The specific layer configuration used in this work is provided in Table 1.

Table 1. U-Net Architecture. The relational PointConv layer in the final decoder block is disabled. We choose $5\,\mathrm{cm}$ as the coarsest resolution, based on the physical size of the real-world objects used in our experiments (approximately 5–$12\,\mathrm{cm}$ along their maximum axis).

Stage	Interaction Block	Resolution Change	Feature Scaling
Input (Pointwise MLP)	–	dense $\rightarrow$ dense	$8 \rightarrow 32$
Encoder	Block 1	dense $\rightarrow$ dense	$32 \rightarrow 32$
	Block 2	dense $\rightarrow$ 2 cm	$32 \rightarrow 64$
	Block 3	2 cm $\rightarrow$ 5 cm	$64 \rightarrow 128$
Bottleneck	Block 4	5 cm $\rightarrow$ 5 cm	$128 \rightarrow 128$
	Block 5	5 cm $\rightarrow$ 5 cm	$128 \rightarrow 128$
Decoder	Block 6	5 cm $\rightarrow$ 2 cm	$128 \rightarrow 64$
	Block 7	2 cm $\rightarrow$ dense	$64 \rightarrow 32$
	Block 8	dense $\rightarrow$ dense	$32 \rightarrow 32$
Prediction Head (Pointwise MLP)	–	dense $\rightarrow$ dense	$32 \rightarrow 3$

2. Training/Test Splits

As described in the main paper, our dataset contains 210 falling cube stack sequences and 292 bowling sequences. We reserve approximately 10% of the data for testing, resulting in 20 cube stack test sequences and 30 bowling test sequences. This leaves 190 cube stack scenes and 262 bowling scenes for training, which users may further divide into training and validation sets as needed. All performance numbers reported in the main paper and this supplementary document are evaluated on the test split of each scenario. The exact list of scene names for each split will be made available on our project webpage.

3. Additional Details on Data Pre-processing

We feed only the Gaussians corresponding to foreground objects into the network. To achieve this, when running Gaussian Splatting (GS) on each frame to generate input Gaussians, we first apply video object segmentation masks to the input RGB images. This ensures that the GS optimization is performed on images containing only the foreground objects of interest. GS may also generate noisy Gaussians along camera rays (i.e., floaters). We filter these out by requiring each Gaussian to be visible from at least three camera views during GS optimization. Additionally, although the background is masked out, GS may still produce Gaussians in background regions. We remove these by checking their rendering contributions within the segmentation mask.

4. Additional Details on Integrating an External Baseline into Our Pipeline

We initially attempted to run GS-Dynamics* on the dense Gaussians produced by Gaussian Splatting [1] for our dataset. However, the model could not be applied directly to these dense inputs, as it does not scale well to high point densities. This limitation likely stems from its use of an ε-ball neighborhood to define per-point connectivity. With dense Gaussian inputs, each point acquires a significantly larger set of neighbors, leading to substantial computational overhead.

To address this issue, [5] select a sparse subset of Gaussians via farthest point sampling and later interpolate the model's predictions back to the original resolution. Following a similar strategy, we apply grid-based downsampling to the original Gaussians—using a grid size of $2\,\mathrm{cm}$ for the cube stack scenario and $4\,\mathrm{cm}$ for the bowling scenario—before feeding them to GS-Dynamics*. We then later apply the Gaussian densification scheme proposed in [5] to each predicted object to recover dense Gaussians. This allows us to evaluate the results using rendering-based metrics, as presented in the main paper.

5. Ablation on Different 4D Gaussian Generation Methods

We compare our approach with dynamic-scene GS methods [3, 4], which can produce 3D Gaussian trajectories for both model input and supervision. Because these methods rely on test-time optimization and are slow to run, we construct a *train-mini* set by selecting 70 scenes from each scenario. We apply the GS baselines to the train-mini set and to the test set (20 falling cube stacks and 30 bowling sequences), train our model on train-mini, and evaluate on the corresponding test split. Each scene is divided into 6-frame clips, and the GS baselines are run on each clip to obtain 6-frame Gaussian trajectories. The first three frames are used as model input and the remaining frames as supervision. Since only short clips are available, long-term tracks are not produced, and we therefore do not use the long-term

Table 2. Ablation results comparing different 4D Gaussian generation methods.

	Bowling				Falling Cube Stacks			
Method	PSNR ↑	SSIM ↑	LPIPS ↓	CD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	CD ↓
Dynamic 3DGS	25.27 ± 0.01	0.975 ± 0.000	0.060 ± 0.000	15.72 ± 0.63	24.21 ± 0.05	0.970 ± 0.000	0.062 ± 0.001	13.71 ± 0.41
4D GS	26.98 ± 0.01	0.980 ± 0.000	0.047 ± 0.000	10.86 ± 0.01	25.27 ± 0.10	0.973 ± 0.000	0.057 ± 0.002	13.12 ± 1.16
Ours	27.57 ± 0.12	0.979 ± 0.001	0.044 ± 0.001	13.98 ± 2.05	25.83 ± 0.07	0.971 ± 0.001	0.055 ± 0.001	10.05 ± 0.62

position metric δ_{avg} here and only report rendering metrics and Chamfer Distance (CD) here.

The ablation results are presented in Table 2. We observe a consistent performance gap between Dynamic 3D GS and our data across both rendering metrics and Chamfer Distance (CD), with our model outperforming the same architecture trained using Dynamic 3D GS. When comparing 4D GS with our data, the performance gap is smaller. However, we find that training with 4D GS in this setting is generally unstable. In practice, we encountered frequent training failures caused by corrupted or degenerate Gaussian reconstructions, and had to manually remove problematic scenes to complete training. Overall, while 4D GS produces visually high-quality renderings, the underlying 3D Gaussian motion is not always physically consistent. In particular, we observe that 4D GS may generate multiple sets of Gaussians for a fast-moving object, and represent it using different Gaussians across frames. This leads to temporally inconsistent trajectories, which can negatively affect dynamics learning.

References

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

[3] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. 1

[4] 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. 1

[5] 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