Motion-Blender Gaussian Splatting for Dynamic Scene Reconstruction

Xinyu Zhang **Haonan Chang** **Yuhan Liu** **Abdeslam Boularias**
{xz653, hc856, yl1834, ab1544}@rutgers.edu
Rutgers University

Abstract: Gaussian splatting has emerged as a powerful tool for high-fidelity reconstruction of dynamic scenes. However, existing methods primarily rely on implicit motion representations, such as encoding motions into neural networks or per-Gaussian parameters, which makes it difficult to further manipulate the reconstructed motions. This lack of explicit controllability limits existing methods to replaying recorded motions only, which hinders a wider application in robotics. To address this, we propose Motion Blender Gaussian Splatting (MBGS), a novel framework that uses motion graphs as an explicit and sparse motion representation. The motion of a graph's links is propagated to individual Gaussians via dual quaternion skinning, with learnable weight painting functions that determine the influence of each link. The motion graphs and 3D Gaussians are jointly optimized from input videos via differentiable rendering. Experiments show that MBGS achieves state-of-the-art performance on the highly challenging *iPhone* dataset while being competitive on *HyperNeRF*. We demonstrate the application potential of our method in animating novel object poses, synthesizing real robot demonstrations, and predicting robot actions through visual planning. The source code and the models are included in the supplementary material. Video demonstrations can be found at `mlzxy.github.io/motion-blender-gs`.

Keywords: Dynamic Scene Reconstruction, Gaussian Splatting

1 Introduction

Reconstructing and modeling dynamic 3D scenes is a fundamental challenge in robot vision. Recent work on 3D Gaussian splatting has made significant progress in capturing dynamic scenes, enabling efficient and high-fidelity reconstruction [1, 2, 3, 4, 5, 6, 7]. Existing Gaussian splatting methods utilize 3D Gaussians to represent geometry and appearance, paired with motion modules that determine the movements of the Gaussians. For instance, 4D-Gaussians [3] and Deformable-GS [8] encode motion implicitly into neural networks. Shape-of-Motion [1] and STG [4] use shallower models that require dense per-Gaussian motion parameters. While these approaches achieve high-fidelity reconstruction, the reconstructed scenes cannot be easily altered or manipulated in simulation due to their implicit motion representation, which limits their use for robot manipulation planning. This lack of direct and explicit controllability restricts existing methods to simply replaying recorded motions.

Therefore, a key question to answer is: can we develop an explicit and sparse motion representation without compromising the ability to reconstruct complex dynamic scenes? To answer this question, we revisit in this work some classical animation techniques [9]. Explicit motion representations, such as deformation graphs [10], harmonic coordinates [11], and cage [12], have been developed to animate objects with diverse motions. However, these classical hand-crafted methods focus on applying manually designed motions to mesh surfaces instead of differentiable models such as 3D Gaussians [13]. Further, these methods are not able to reconstruct motion or geometry from images.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

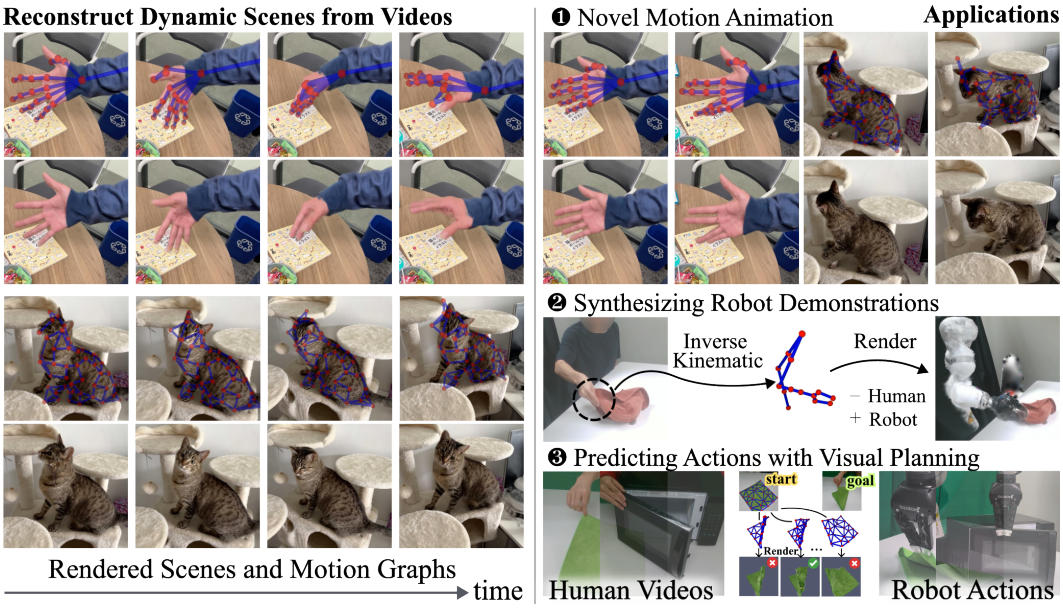


Figure 1: **Capabilities of Our Framework.** Our method reconstructs and renders dynamic scenes into 3D Gaussians and motion graphs from input videos. The learned motion graphs for a hand and cat are shown with their corresponding rendered scenes (left). Our approach enables three key applications (right): ❶ Novel pose animation through motion graph editing, ❷ Robot demonstration synthesis by using robot kinematic chains as motion graphs, and ❸ Predicting robot actions by simulating graph movements to minimize the difference between rendered and goal images.

Inspired by classical mesh animation techniques, we propose using kinematic trees and deformable graphs as motion representations for Gaussian splatting-based reconstruction. Kinematic trees are well-suited for capturing the motions of articulated objects, such as human bodies or robots, while deformable graphs, free from topological constraints, are ideal for modeling non-rigid object deformations. We collectively refer to both kinematic trees and deformable graphs as *motion graphs*. Motions of graph links are propagated to individual Gaussians through dual quaternion skinning [14]. The influence of each graph link on a Gaussian is predicted by a learnable weight painting function. The graphs are initialized from point clouds and 2D keypoints at a canonical frame. Both the Gaussians and motion graphs are optimized end-to-end jointly from videos via differentiable rendering. We term our approach Motion-Blender Gaussian Splatting (MBGS).

Our main contributions can be summarized as follows. (1) We introduce a new dynamic Gaussian splatting framework based on explicit and sparse motion graphs, which allows for straightforward robot manipulation planning in reconstructed scenes. Gaussian motions are predicted by blending the link motions. (2) We propose two types of parametric motion graphs—kinematic trees and deformable graphs—and a learnable weight painting function based on Gaussian kernels, along with optimization details in Sec. A.2. Our method learns both 3D Gaussians and motion graphs jointly. (3) Compared with state-of-the-art, our method outperforms Shape-of-Motion on the challenging iPhone dataset [15], and achieves competitive performance with 4DGaussians on HyperNerf [16]. Further, we demonstrate the applications of our approach on animating novel object motions, synthesizing real robot demonstrations, and predicting robot actions through visual planning. This can lead to significant efficiency improvement in gathering training datasets for robot learning [17].

2 Method

Preliminaries. Gaussian splatting represents a static 3D scene explicitly with a set $\mathcal{G}$ of 3D Gaussians. Each 3D Gaussian $g \in \mathcal{G}$ is parameterized by its pose $\mathbf{p} \in \mathrm{SE}(3)$, scale $\mathbf{s} \in \mathbb{R}^3$, opacity $o \in \mathbb{R}$ and color $\mathbf{c} \in \mathbb{R}^3$. 2D images can be rendered from $\mathcal{G}$ by blending the colors of overlapping

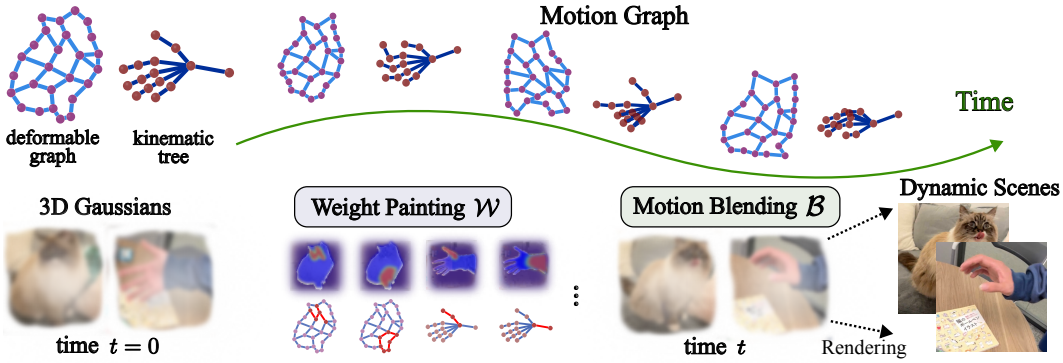


Figure 2: **Motion Blender Gaussian Splatting.** Our framework explicitly represents motion using sparse dynamic graphs. Static 3D Gaussians are associated with the graphs through learnable weight painting. Then, link-wise motions are propagated to the Gaussians through motion blending with dual quaternion skinning. We employ two motion graph types: kinematic trees, ideal for capturing articulated structures like human bodies, and deformable graphs, designed for modeling non-rigid deformations in soft objects. The parameters of the motion graph, weight painting functions, and 3D Gaussians are jointly optimized, end-to-end, via differentiable rendering.

Gaussians along the ray direction of each pixel. Dynamic Gaussian splatting extends each Gaussian by making it time-dependent, g_t, where $t \in [0, T-1]$ and T represents the number of frames in the source video. The common practice is to make the pose time-dependent, $\mathbf{p}_t$, and keep the rest static. $\mathbf{p}_t$ is often obtained from a dynamic model f as in Eq. 1, where θ denotes the model's parameters.

$$\mathbf{p}_t = f(\mathbf{p}_0, t; \theta), \text{where } t > 0 \tag{1}$$

f is often implemented as a neural net, *e.g.*, a deformation field network [3, 8], or as a shallow model such as motion coefficients or polynomials [18, 4, 1] that require per-Gaussian motion parameters.

2.1 Motion Blender Gaussian Splatting

Our framework is designed based on two key ideas that set it apart from existing works. 1 Represent motions with a sparse structure: Instead of associating dense motion parameters to each Gaussian (often numbering in the hundreds of thousands), we use a motion graph with far fewer parameters (often less than 100). The sparse link-wise motions are then propagated to each Gaussian through motion blending. 2 Represent motions explicitly: Instead of encoding motions implicitly into neural networks, our motion graph is an explicit kinematic model. This allows a straightforward visualization and manipulation of motions in the 3D space, such as transferring motion patterns from one object to a new object, or creating novel animations through motion graph editing.

Formally, we define the motion graph $G_t = (\mathcal{J}_t, \mathcal{L})$ at time t as a set of joints $\mathcal{J}_t$ with a set of edges $\mathcal{L}$, where $\max(|\mathcal{J}_t|, |\mathcal{L}|) \ll |\mathcal{G}|$. Each link $l \in \mathcal{L}$ is defined as $l = (s_l, e_l)$ and consists of the start and end joint indexes $s_l, e_l \in [1, |\mathcal{J}_t|]$. The positions of all the links at time t are arranged as a matrix $X_{\mathcal{L},t} \in \mathbb{R}^{|\mathcal{L}| \times 6}$, where each row corresponds to the 3D coordinates of the start and end joints for a link in $\mathcal{L}$. Similarly, $\mathbf{x}_t \in \mathbb{R}^3$ denotes the position of a Gaussian with pose $\mathbf{p}_t$ at time t. We assume both the edges $\mathcal{L}$ and the number of joints $|\mathcal{J}_t|$ stay static, and only joint positions undergo motions over time. Let $P_{\mathcal{L},t} \in \mathrm{SE}(3)^{|\mathcal{L}|}$ be the poses of all the links at time t. Let $\mathbf{p}_0$ be the parameter that describes the initial pose of a Gaussian, then $\mathbf{p}_t$ is given by Eq. 2.

$$\mathbf{p}_t = \mathcal{B}\big(\mathcal{R}\left(P_{\mathcal{L},0}, P_{\mathcal{L},t}\right), \mathcal{W}\left(\mathbf{x}_0, X_{\mathcal{L},0}\right)\big) \cdot \mathbf{p}_0 \tag{2}$$

$\mathcal{W} : \mathbb{R}^3 \times \mathbb{R}^{|\mathcal{L}| \times 6} \mapsto \Delta^{|\mathcal{L}|-1}$ denotes the *weight painting function*, where $\Delta^{|\mathcal{L}|-1}$ represents the probability vector space defined as $\{\mathbf{w} \in \mathbb{R}_+^{|\mathcal{L}|} \mid \sum_{i=1}^{|\mathcal{L}|} \mathbf{w}_i = 1\}$. This function $\mathcal{W}$ estimates the level of influence of each link $l \in \mathcal{L}$ on each Gaussian based on their relative positions at $t = 0$.

$\mathcal{R} : \mathrm{SE}(3)^{|\mathcal{L}|} \times \mathrm{SE}(3)^{|\mathcal{L}|} \mapsto \mathrm{SE}(3)^{|\mathcal{L}|}$ returns relative rigid transforms between two sets of 3D poses. Specifically, $\mathcal{R}\left(P_{\mathcal{L},0}, P_{\mathcal{L},t}\right)$ indicates the SE(3) movement from time 0 to time t of each link in $\mathcal{L}$.

$\mathcal{B} : \mathrm{SE}(3)^{|\mathcal{L}|} \times \Delta^{|\mathcal{L}|-1} \mapsto \mathrm{SE}(3)$ represents the *motion blending function*, which computes the per-Gaussian motion by propagating the link-wise movements $\mathcal{R}\left(P_{\mathcal{L},0}, P_{\mathcal{L},t}\right)$ to each Gaussian based on the weights assigned by $\mathcal{W}$ at $t = 0$.

We implement $\mathcal{B}$ using dual quaternion skinning (DQS) [14]. DQS represents transformations using dual quaternions and computes weighted averages in a way that guarantees the resulting transformation remains valid in SE(3) space. The link-wise movements computed by $\mathcal{R}$ are analytically determined, as the relative SE(3) transforms can be straightforwardly derived. Therefore, our motion blender framework hinges on two key design choices: (1) Modeling graph motions by representing their temporal evolution through link poses $P_{\mathcal{L},t}$. (2) Defining a weight painting function $\mathcal{W}$ that accurately captures the influence of each graph link on a Gaussian's motion. We address the former by presenting two types of parametric motion graphs in Sec. 2.2. The latter, the design of $\mathcal{W}$, is detailed in Sec. 2.3. The overall motion-blender Gaussian splatting framework is illustrated in Fig. 2.

2.2 Motion Graph Representation

To predict $P_{\mathcal{L},t}$, the link poses at time t, we use a parameterized function $\mathcal{P}_{\mathcal{L}}(\theta, \phi) \in \mathrm{SE}(3)^{|\mathcal{L}|}$. Here, θ is the graph parameters that determine the link poses at a given instant, and ϕ is the time-independent graph parameters. By modeling θ as a time-varying sequence $(\theta_t)_{t=0}^{T-1}$, the link poses at time t are defined as $P_{\mathcal{L},t} = \mathcal{P}_{\mathcal{L}}(\theta_t, \phi)$, where θ_t describes the kinematic state of the graph at time t. Two types of motion graphs, kinematic trees and deformable graph, are shown in Fig. 3.

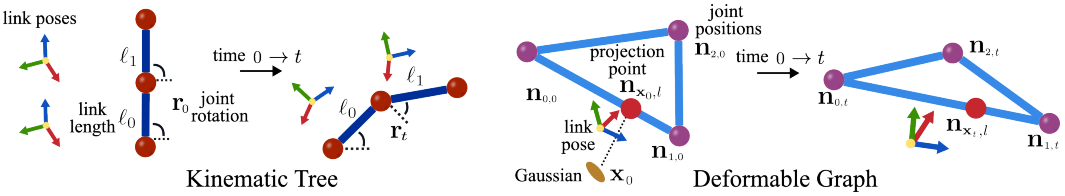


Figure 3: **Motion Graphs.** A kinematic tree (left) uses time-independent link lengths ℓ and dynamic joint rotations $\mathbf{r}_t \in \mathrm{SO}(3)$. Link poses (shown as colored coordinate axes) in world coordinates are computed through forward kinematics. A deformable graph (right) employs free-form topology parameterized by joint positions $\{\mathbf{n}_{i,t}\}$ and has non-rigid link deformations. Rigid per-link poses are obtained relative to each Gaussian position $\mathbf{x}_0$ and look-at transformation as in Eq. 3 and Eq. 4.

Kinematic Tree. A kinematic tree is a hierarchical, acyclic graph with a root joint. This representation is ideal for capturing the kinematic chains of articulated objects such as human bodies, robot arms, or other multi-joint systems. We parameterize the tree using $\theta = (\mathbf{r}, \mathbf{X}), \phi = (\ell_i)_{i=1}^{L}$, where $\ell_i \in \mathbb{R}^+$ denotes the length of the i-th link, $\mathbf{r} \in \mathrm{SO}(3)^{|\mathcal{J}_t|-1}$ represents the joint rotations, and $\mathbf{X} \in \mathrm{SE}(3)$ denotes the pose of the root joint. Then $\mathcal{P}_{\mathcal{L}}(\theta_t, \phi)$ can be implemented via a forward kinematics algorithm [19], which propagates transformations from the root node through the tree, compounding rotations and translations across joints to derive every link pose. The forward kinematic procedure is differentiable, which enables the learning of θ and ϕ through back-propagation.

Deformable Graph. Unlike the kinematic tree, a deformable graph imposes no topological constraints, allowing the joints to move freely in the 3D space. This representation is ideal for modeling non-rigid deformations and surface variations in soft objects. We parameterize the graph as $\theta = (\mathbf{n}_j \in \mathbb{R}^3)_{j=1}^{|\mathcal{J}_t|}$, where $\mathbf{n}_j$ denotes the 3D position of the j-th joint. However, the absence of topological constraints allows links between joints to stretch or compress, making it impossible to describe link poses using rigid transformations. To address this, we project each Gaussian $\mathbf{x}$ in the first frame on each link in the graph, and track the positions of the projected points in the remaining frames. Given each Gaussian $\mathbf{x}$, the matrix of the poses of all the links, denoted as $P_{\mathcal{L}}(\theta, \mathbf{x})$, is given by the 3D positions of the closest-point projections of $\mathbf{x}$ on the links in addition to the 3D directions of the links. Specifically, for each link l with start joint $\mathbf{n}_{s_l,0}$ and end joint $\mathbf{n}_{e_l,0}$ at $t = 0$, we define

Figure 4: **Learned Motion Graphs and Weight Paintings.** The first row overlays the learned motion graphs on the images. The second row shows painted weights (red) of graph links (green).

$\mathbf{n}_{\mathbf{x}_0,l} \in \mathbb{R}^3$ as the point on l that has the minimal distance to $\mathbf{x}_0$. We refer $\mathbf{n}_{\mathbf{x}_0,l}$ as the projection point at $t = 0$. Next, we define the projection point $\mathbf{n}_{\mathbf{x}_t,l}$ at time t in Eq. 3.

$$\mathbf{n}_{\mathbf{x}_t,l} = \mathbf{n}_{s_l,t} + \frac{|\mathbf{n}_{\mathbf{x}_0,l} - \mathbf{n}_{s_l,0}|}{|\mathbf{n}_{s_l,0} - \mathbf{n}_{e_l,0}|}(\mathbf{n}_{s_l,t} - \mathbf{n}_{e_l,t}) \tag{3}$$

In other words, the projection point $\mathbf{n}_{\mathbf{x}_t,l}$ moves proportionally to the stretching of link l. Therefore, the link poses can be defined using the projection point $\mathbf{n}_{\mathbf{x}_t,l}$, decoupling from non-rigid stretching deformations. Thus, the link pose $P_{\mathcal{L}}(\theta_t, \mathbf{x}_t)$ is derived via look-at transformation in Eq. 4.

$$P_{\mathcal{L}}(\theta_t, \mathbf{x}_t) = \{\mathscr{A}(\mathbf{n}_{\mathbf{x}_t,l}, \text{ray}(\mathbf{n}_{s_l,t}, \mathbf{n}_{e_l,t})), \forall l \in \mathcal{L}\} \tag{4}$$

where $\text{ray}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{b}-\mathbf{a}}{|\mathbf{b}-\mathbf{a}|}$ is the unit direction vector from $\mathbf{a}$ to $\mathbf{b}$. The look-at transformation $\mathscr{A} : \mathbb{R}^3 \times S^2 \mapsto \text{SE}(3)$ maps the viewing position and direction to a $\text{SE}(3)$ pose [20]. Note that look-at transformation requires defining an up-direction for each link. We organize the $|\mathcal{L}|$ links into $\frac{|\mathcal{L}|}{2}$ triangles and use their face normals as up-directions. Remark that there is a key distinction between our deformable graph and the deformation graph used in Dynamic Fusion [21]. Dynamic Fusion attaches SE(3) poses to joints only. Our method represents each joint as an $\mathbb{R}^3$ position and derives SE(3) poses at links, which simplifies motion-graph manipulation.

Graph Connectivity $\mathcal{L}$ Initialization. The last problem to address is determining the graph connectivity $\mathcal{L}$, which defines the links between joints as integer pairs $l = (s_l, e_l)$, where $s_l, e_l \in [1, |\mathcal{J}_t|]$ denote the indices of the start and end joints. The connectivity $\mathcal{L}$ is treated as a fixed parameter, neither receiving gradients nor changing over time. For kinematic trees, connectivity is derived from domain-specific priors, such as lifting 2D human skeletons to 3D or extracting from robot models. For deformable graphs, we initialize $\mathcal{L}$ by randomly sampling and connecting points from a point cloud, using farthest point sampling [22] to ensure uniform coverage of the object's surface, as illustrated in Fig. 6 and detailed in Appendix A.2.

2.3 Weight Painting Function

To estimate the influence of each graph link on a Gaussian, the weight painting function is given as: $\mathcal{W}(\mathbf{x}_0, X_{\mathcal{L},0}) = \text{softmax}(\{K(\mathbf{x}_0, X_{\mathcal{L},0,i}) \mid \forall i \in [1, |\mathcal{L}|]\})$, where $\mathbf{x}_0$ is the initial position of a Gaussian, and $X_{\mathcal{L},0,i} \in \mathbb{R}^6$ are the initial positions of start and end joints of the i-th link. The kernel function $K : \mathbb{R}^3 \times \mathbb{R}^6 \to \mathbb{R}$ measures the affinity between a Gaussian and a link and is defined as:

$$K(\mathbf{x}_0, X_{\mathcal{L},0,i}) = \exp\left(-\gamma \cdot \text{dist}(\mathbf{x}_0, X_{\mathcal{L},0,i})\right) \tag{5}$$

where γ is a learnable parameter that controls the kernel radius, and $\text{dist}(a, b)$ is the distance between point a to line segment b. By learning distinct γ at each link, the motion graph's influence regions collectively span the object's surface. At $t = 0$, 3D Gaussians reconstruct objects' initial geometry. The static reconstruction is bonded to the dynamic motion graph through weight painting. The painted weights are subsequently used to propagate motions for $t > 0$, for dynamic reconstruction.

3 Experiment

3.1 Settings

Datasets. We evaluate our method on two real-world datasets: the highly challenging iPhone dataset [15] from [1] and the HyperNeRF [16] vrig dataset. The iPhone dataset consists of 12 scenes,

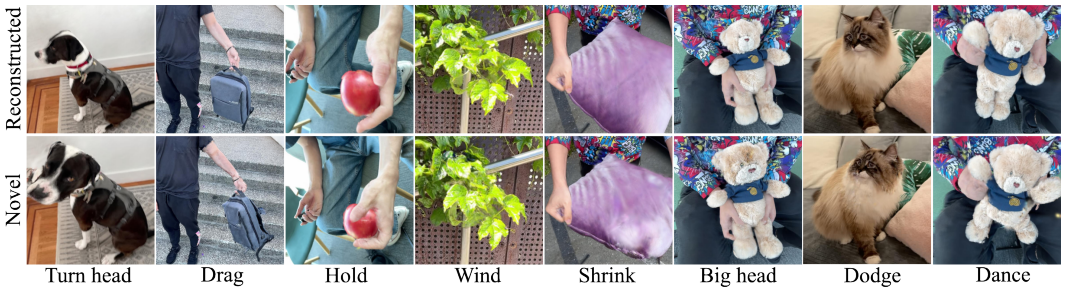
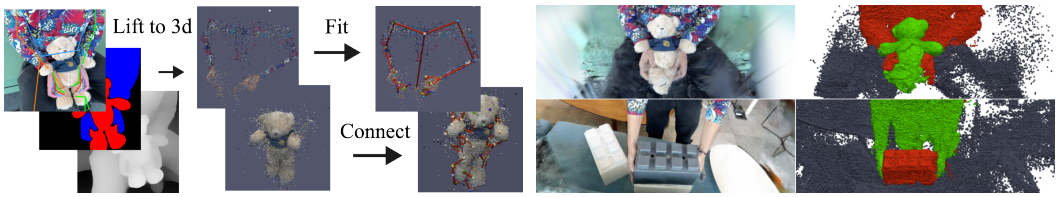


Figure 5: **Novel Poses from Motion Graph Manipulation.** The first row shows images of scenes reconstructed from observations. By applying new control actions on the motion graphs and propagating changes to the Gaussians, novel unseen poses are imagined and rendered in the second row.



(a) Motion Graph Initialization (b) Instance-level Reconstruction

Figure 6: We initialize motion graphs at the canonical frame ($t = 0$) using instance segmentation masks from Grounding SAM2 [23, 24] and 2D human skeletons estimated by SAPIENS [25]. Furthermore, our framework enables per-instance reconstruction, where Gaussians are explicitly grouped to maintain accurate instance geometry — a capability previously unexplored in existing literature of dynamic Gaussian splatting. More optimization details are provided in Appendix A.2.

including 5 multi-camera (MV) and 7 single-camera (SV) scenes, each captured at 960×720 resolution. The MV scenes are captured using one hand-held moving camera and two fixed cameras at novel angles, where the hand-held video is used for training and the fixed cameras for quantitative evaluation. The SV scenes, captured solely with a hand-held camera, are used for qualitative visualization. The HyperNeRF vrig dataset consists of 4 scenes, each captured at 960×540 resolution using two cameras mounted on a rig kit with strong camera motions. Models are trained on a subset of frames and evaluated on the remaining frames.

Baselines. On the iPhone dataset, we compare against Shape-of-Motion [1]. For HyperNeRF, we compare against 4DGaussians [3]. Shape-of-Motion and 4DGaussians are the state of the art on their respective datasets. We use the 2D track loss proposed in [1], in addition to L1 RGB loss.

Metrics. We report peak-signal-to-noise ratio (PSNR), structural similarity index (SSIM) [26] and learned perceptual image patch similarity (LPIPS) [27]. Since traditional metrics such as PSNR and SSIM are sensitive to minor misalignments and often favor blurry images over sharp ones [16], we adopt LPIPS as our primary metric.

3.2 Results

Table 1: Novel view rendering on the highly challenging iPhone dataset [1]. LPIPS more accurately reflects perceptual quality.

Method	LPIPS ↓	PSNR ↑	SSIM ↑
T-NeRF [15]	0.55	15.6	0.55
HyperNeRF [16]	0.51	15.99	0.59
DynIBaR [28]	0.55	13.41	0.48
Deformable-GS [8]	0.66	11.92	0.49
4DGaussians [3]	0.56	13.42	0.49
Shape-of-Motion [1]	0.39	16.67	**0.65**
Ours	**0.37**	**16.79**	**0.65**

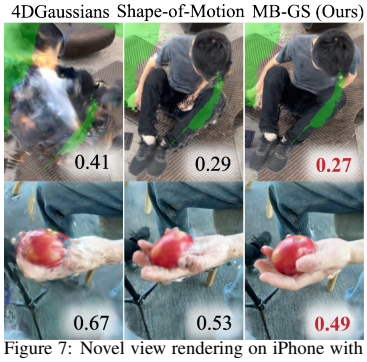


Figure 7: Novel view rendering on iPhone with best LPIPS in red. More results in Fig. A4.

Table 2: HyperNerf [16]. Our method performs competitively, closely matching SoTA in the key LPIPS metric.

	LPIPS ↓	PSNR ↑	SSIM ↑
4DGaussians [3]	0.36	**25.19**	**0.68**
Shape-of-Motion [1]	**0.34**	21.01	0.54
Ours	0.35	20.60	0.54

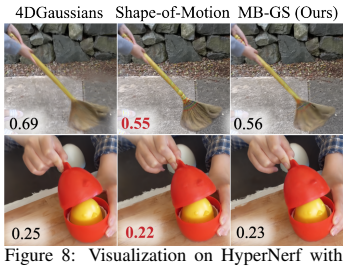


Figure 8: Visualization on HyperNerf with best LPIPS in red. More results in Fig. A5

Tab. 1 presents our quantitative results on the iPhone benchmark, where our method outperforms the state-of-the-art Shape-of-Motion by 0.02 LPIPS. Fig. A4 provides a qualitative comparison on the five MV scenes with 4DGaussians [3] and Shape-of-Motion [1], demonstrating that our method renders sharper, more complete, and perceptually higher-quality novel views. While 4DGaussians achieves comparable PSNR and SSIM scores, its rendering quality is notably inferior. Fig. 4 visualizes the learned motion graphs and weight paintings. Training on the iPhone dataset converges within 10 to 30 hours on a 40GB A100 GPU, depending on scene complexity. For the teddy bear scene with 2M Gaussians—using a deformable graph for the bear and a kinematic tree for the human—our method achieves a rendering speed of 18 FPS, and 25 FPS if motion graphs for each frame are pre-computed and cached. On smaller scenes, such as the chicken toy in HyperNeRF with 300K Gaussians, the rendering speed increases to 32 FPS without caching and 46 FPS with caching.

Tab. 2 presents our quantitative results on the HyperNeRF vrig benchmark, with qualitative comparison in Fig. A5. While 4DGaussians, the SoTA on HyperNeRF, achieves higher PSNR and SSIM, our method achieves a comparable or better rendering quality and LPIPS on the chicken, 3D printer, and broom scenes (rows 2 to 4 in Fig. A5). Notably, 4DGaussians achieves a better score on the 3D printer scene, but our method produces clearer results, accurately rendering the text on the printer motor, unlike the blurry output from 4DGaussians. However, on the peel-banana scene, our method produces lower-quality results than 4DGaussian. We further discuss our limitations in Sec. 5.

3.3 Applications

Novel Pose Animation. Fig. 5 shows images of objects in novel poses that were not seen in the training videos. After reconstructing dynamic scenes, we keep the Gaussians fixed and modify the motion graph starting from a sampled frame. Specifically, we adjust joint rotation angles in a kinematic tree or manipulate the positions of subsets of joints in a deformable graph. These edits are performed interactively using a custom visual editor that we built on VISER [29]. Once the edits are applied, we propagate the updated graph link poses to the Gaussians and render the scene from the same camera viewpoint as the sampled frame. To create short animations, we interpolate joint angle or position changes incrementally and render the scene frame-by-frame.

Fig. 5 shows that our method extends beyond reconstructing observed motions, enabling the creation of novel imagined object motions while maintaining high rendering quality. Recent advances in controllable image and video editing predominantly rely on large diffusion models [30, 31, 32]. In contrast, our method introduces a novel approach by operating directly in the 3D space, enabling fast and precise control over object poses without the need for neural networks.

Robot Demonstration Synthesis. Imitation learning enables robots to perform complex tasks by learning from robot demonstrations [33]. Unlike human videos, collecting robot demonstrations requires labor-intensive teleoperation [34]. In Fig. 9, we demonstrate a prototype of our method for synthesizing robot demonstrations from human videos. We reconstruct scenes from human and robot videos independently. The human videos feature an operator performing pick-and-place and cloth folding tasks. The robot videos feature a Kuka iiwa robot that moves randomly. We use the robot's kinematic chain as the motion graph. We then remove the human Gaussians and add the robot Gaussians to the workspace. Keyframes are manually selected, with hand positions adjusted and mapped to robot gripper poses using our visual editor. Gripper poses between keyframes are interpolated, and converted into joint angles of the robot's kinematic chain using inverse kinematics.

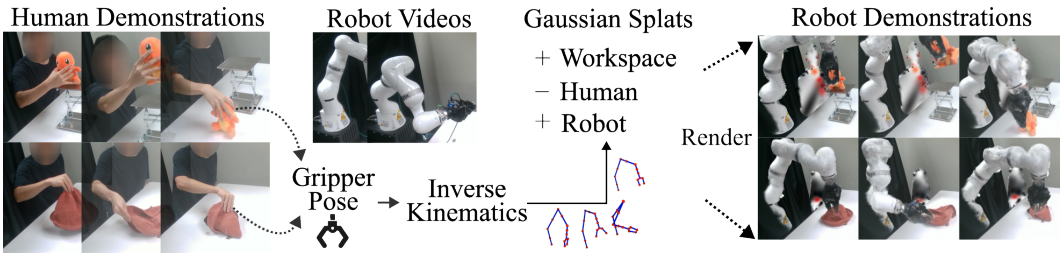


Figure 9: **Robot Demonstration from Human Videos.** We reconstruct Gaussians from human and robot videos, with robot's pre-defined kinematic chain as motion graph. We then remove human's Gaussians and add robot's to workspace. Next, we animate the robot motion graph using inverse kinematics derived from hand poses. This renders videos of a robot arm mimicking human actions.

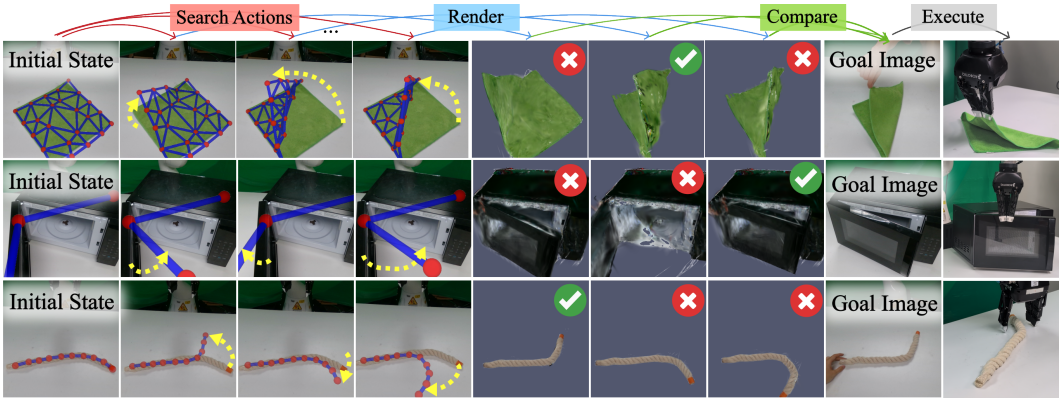


Figure 10: **Robotic Action Prediction via Visual Planning.** We reconstruct object Gaussians (cloth, microwave, rope) from human videos, simulate potential motion graph trajectories, and render the resulting object images. The optimal trajectory is selected by maximizing PSNR between rendered images and the target goal image. Key joint trajectories (yellow) are mapped to end-effector positions. Videos of our robot experiments can be found at `anonymous368.github.io/motion-blender`.

The chain is then used as the robot motion graph for view rendering. This generates a video where the robot replaces the human, completing the task as demonstrated. While some artifacts and unrealistic interactions still exist, this demonstrates the potential of our method for robotic data synthesis. The videos are obtained with minimal effort, and can be used for learning vision-based policies.

Robot Visual Planning. Predicting action outcomes is a fundamental human capability that enables complex manipulation skills, such as manipulating objects until a certain goal is reached [35]. While recent work has explored video generation models to incorporate this ability for robotic manipulation [36, 37], these approaches often require complicated pipelines. In Fig. 10, we present a simple visual planning prototype for goal-conditioned manipulation of deformable (cloth and rope) and articulated (microwave door) objects. We first reconstruct object-specific Gaussians from human videos. For each test scene, the motion graph is learned by minimizing the L1 rendering loss from our reconstructed Gaussians, while keeping Gaussians and weights fixed. This process typically converges within a minute. We then simulate various graph trajectories and render the resulting object images. The optimal trajectory is selected by maximizing PSNR with the provided target goal image. In summary, our method learns a dynamic and photorealistic model of the manipulated object *on the fly*, using a very small number of frames, and performs planning in simulation with the learned model to select optimal actions. The reward function is simply provided in the form of a goal image. This prototype demonstrates the potential of our method for developing more data-efficient model-based robot learning solutions. More experiment details are provided in Appendix A.3.

4 Final Remarks

We conducted ablation studies analyzing motion graph sizes and regularization strategies. Ablation studies, related work discussions, more visualizations and optimization details are included in Appendix A due to space limit. We discuss the limitations and possible future directions in Sec. 5.

5 Limitations and Future Directions

Visual Artifacts on Novel Poses. Fig. 11 (b) demonstrates failure cases in animating novel object poses, where attempting to turn the cat's head and rotate the windmill's badge introduces non-negligible visual artifacts. Visual artifacts can also be found in Fig. 9 and Fig. 10 of our robot experiments. We believe the key reason is that, unlike meshes, Gaussians lack an explicit *surface representation*. This allows each individual Gaussian to deform arbitrarily when motion graphs are applied. This can cause some Gaussians to deviate from object surfaces, particularly for unseen motions. Recent advances like 2D-GS [38], which directly formulates Gaussians on 3D surfaces, offer promising solutions to this limitation.

(a) Ungrounded Motion Graphs (b) Visual Artifacts on Novel Poses (c) Lost of Reflection

Figure 11: **Failure Cases.** Visualization of imperfect learned motion graphs (a), failure cases of novel pose editing (b), and failure cases of reconstructing reflective surfaces (c).

Ungrounded Motion Graphs. Fig. 11 (a) demonstrates imperfect learned motion graphs that are not accurately grounded to the object geometry. Top left: the hand contains intricate kinematic structures but only occupies a small region (less than 80×80 pixels in a 720×920 image), making it difficult to capture fine details. Top right: incomplete motion graph coverage on the cat's right front leg and joints protruding beyond the cat body. Bottom left: the pillow is squeezed but the motion graph fails to sufficiently deform. Bottom right: the kinematic tree does not accurately align with the microwave door.

We believe the key reason is that Gaussian splatting reconstructs scenes purely from visual observations, without incorporating structural or physical priors. For instance, Fig. 12 illustrates a failure case in rope manipulation. While the reconstructed motion graph appears accurate in 2D image space, a 3D inspection reveals that the rope's head tilts below the table—an incorrect reconstruction undetectable from visual observation alone. Since the robot's end-effector pose is derived from the motion graph's joints and links, this leads to an erroneous orientation in the planned trajectory. However, recovering sparse structures from only a few videos is a highly under-constrained optimization problem. A promising direction is to incorporate semantic or physical priors from foundation models to improve geometric consistency.

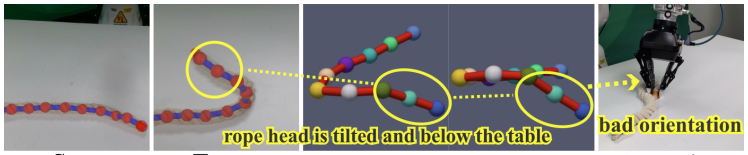


Figure 12: A failure case of incorrect gripper orientation caused by an erroneous motion graph. The head of the rope motion graph is misaligned, tilting downward below the table surface.

Fast Camera and Object Motions. Fig. 13 shows failure cases when reconstructing fast-moving objects. The left figure compares our MB-GS with Shape-of-Motion on adjacent frames from the HyperNerf peel-banana scene, which is captured with strong camera motions and objects (the hand and banana) frequently entering and exiting the view. In this setting, MB-GS produces inconsistent results across frames such as a shaking hand. The right figure compares MB-GS reconstructions of a microwave with doors moving at different speeds. We observe that reconstruction quality degrades significantly when the door moves rapidly. Dynamic reconstruction under large motions remains a longstanding challenge [3]. Shape-of-Motion attempts to address this by using 2D tracks [39] as motion guidance. However, 2D tracks are often unreliable, especially under orientation changes, such as when the microwave doors rotate and original tracks on the door surface are lost. Another limitation is that MB-GS learns motion graph parameters independently at each frame, making it less effective at leveraging temporal continuity across frames. A promising direction is to still follow the motion blender framework but introduce neural networks that predict the deformation of motion graphs over time, preserving the manipulability of the representation while improving learning capacity.

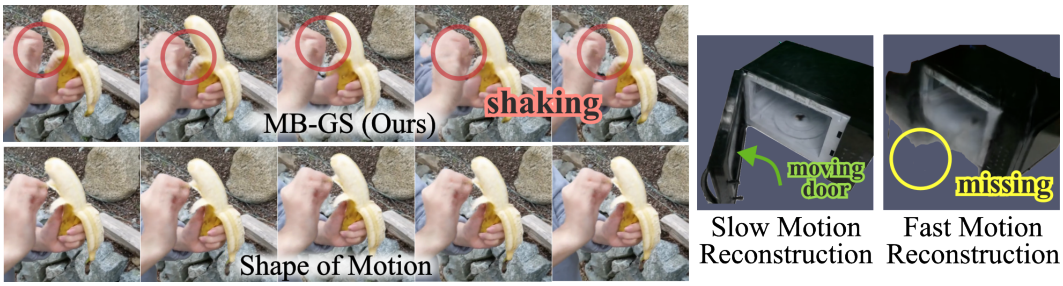


Figure 13: **Failure Cases on Fast-Moving Objects.** Visualization of shaking artifacts from strong camera motion (left), and reconstruction quality comparison of our MB-GS for the same microwave with doors moving at different speeds (right).

Robotic Applications. In the following, we highlight two promising directions that are specific to the robotic applications of our method MB-GS.

- *Motion Graphs and Physics Simulation.* Our framework synthesizes robotic demonstrations for tasks like grasping toys and folding clothes (Fig. 9), and performs planning in graph space to reach goal configurations (Fig. 10). However, our current motion graph representation lacks physical awareness. This is evident in Fig. 9 (second row), where the gripper penetrates the cloth in a physically unrealistic manner. Additionally, the graph-space planning is limited to simple shape (2D rectangle) and basic kinematic chain (chain-like graph), whose motion patterns can be reasonably approximated with geometric heuristics.

  Our sparse motion graph design naturally extends to physics-based simulation frameworks. For instance, MuJoCo [40] models deformable objects using flex elements (triangles in 2D, tetrahedra in 3D), which aligns closely with our deformable graph representation. To further bridge the gap, it is also possible to, first, incorporate not only link poses but also triangular face poses to guide Gaussian motion, leveraging projection points analogous to Fig. 3; second, support hybrid deformable and kinematic structures through a more systematic engineering integration. These extensions would significantly improve simulation compatibility for both articulated and deformable objects.

  In doing so, instead of learning joint positions or rotation angles as in our current approach, it is possible to learn MuJoCo's physical parameters through differentiable simulation: simulating the structure, rendering images, and back-propagating through the physics engine. This would be particularly feasible in controlled environments with clear object segmentation, enabling more powerful digital twins that handle deformable and articulated objects — a capability lacking in current rigid-body-focused digital twins [41].

- *Modeling of Light Sources and Reflective Surfaces.* Robotic workspaces typically contain strong light sources, and robot arms often have reflective surfaces. However, as shown in Fig. 11, our reconstructed robot surfaces appear blurry and lose their reflectivity. This occurs because Gaussian splatting does not account for lighting effects during reconstruction. While this limitation is acceptable for vanilla Gaussian splatting in static scenes (where lighting effects remain mostly constant), it becomes problematic for dynamic scenes. In such cases, surfaces under varying lighting conditions appear to move constantly, leading to blurry reconstructions. Furthermore, workspace lighting affects not only robot reconstruction but also object reconstruction, as objects may be reflective or cast different shadows when illuminated from various angles. We find that increased lighting generally results in blurrier reconstructed objects. While this is a fundamental limitation of current Gaussian splatting methods rather than specific to our approach, it highlights the need for reflection-aware Gaussian splatting techniques to improve robotic applications.

References

[1] Q. Wang, V. Ye, H. Gao, J. Austin, Z. Li, and A. Kanazawa. Shape of motion: 4d reconstruction from a single video. *arXiv preprint arXiv:2407.13764*, 2024.

[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

[3] G. Wu, T. Yi, J. Fang, L. Xie, X. Zhang, W. Wei, W. Liu, Q. Tian, and X. Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20310–20320, 2024.

[4] Z. Li, Z. Chen, Z. Li, and Y. Xu. Spacetime gaussian feature splatting for real-time dynamic view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8508–8520, 2024.

[5] Z. Yang, H. Yang, Z. Pan, and L. Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. *arXiv preprint arXiv:2310.10642*, 2023.

[6] 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*, pages 4389–4398, 2024.

[7] Y.-H. Huang, Y.-T. Sun, Z. Yang, X. Lyu, Y.-P. Cao, and X. Qi. Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4220–4230, June 2024.

[8] Z. Yang, X. Gao, W. Zhou, S. Jiao, Y. Zhang, and X. Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20331–20341, 2023. URL https://api.semanticscholar.org/CorpusID:262466218.

[9] D. L. James and C. D. Twigg. Skinning mesh animations. *ACM Transactions on Graphics (TOG)*, 24(3):399–407, 2005.

[10] R. W. Sumner, J. Schmid, and M. Pauly. Embedded deformation for shape manipulation. In *ACM siggraph 2007 papers*, pages 80–es. 2007.

[11] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. *ACM transactions on graphics (TOG)*, 26(3):71–es, 2007.

[12] J. R. Nieto and A. Susín. Cage based deformations: a survey. In *Deformation Models: Tracking, Animation and Applications*, pages 75–99. Springer, 2012.

[13] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org.

[14] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, 2007.

[15] H. Gao, R. Li, S. Tulsiani, B. Russell, and A. Kanazawa. Monocular dynamic view synthesis: A reality check. *Advances in Neural Information Processing Systems*, 35:33768–33780, 2022.

[16] K. Park, U. Sinha, P. Hedman, J. T. Barron, S. Bouaziz, D. B. Goldman, R. Martin-Brualla, and S. M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021.

[17] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, A. Lee, K. Fang, C. Finn, and S. Levine. Bridgedata v2: A dataset for robot learning at scale. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 1723–1736. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/walke23a.html.

[18] Y. Lin, Z. Dai, S. Zhu, and Y. Yao. Gaussian-flow: 4d reconstruction with dynamic 3d gaussian particle. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21136–21145, 2024.

[19] S. Kucuk and Z. Bingul. *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006.

[20] Scratchapixel. Framing: The look-at function. https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function/framing-lookat-function.html, 2025. Accessed: 2025-02-20.

[21] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015.

[22] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE transactions on image processing*, 6(9):1305–1315, 1997.

[23] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, Z. Zeng, H. Zhang, F. Li, J. Yang, H. Li, Q. Jiang, and L. Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024.

[24] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer. Sam 2: Segment anything in images and videos, 2024. URL https://arxiv.org/abs/2408.00714.

[25] R. Khirodkar, T. Bagautdinov, J. Martinez, S. Zhaoen, A. James, P. Selednik, S. Anderson, and S. Saito. Sapiens: Foundation for human vision models. *arXiv preprint arXiv:2408.12569*, 2024.

[26] Z. 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.

[27] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[28] Z. Li, Q. Wang, F. Cole, R. Tucker, and N. Snavely. Dynibar: Neural dynamic image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4273–4284, 2023.

[29] N. S. Group. Viser: Web-based 3d visualization + python. https://github.com/nerfstudio-project/viser, 2023.

[30] D. Ceylan, C.-H. P. Huang, and N. J. Mitra. Pix2video: Video editing using image diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 23206–23217, 2023.

[31] Y. Shi, C. Xue, J. H. Liew, J. Pan, H. Yan, W. Zhang, V. Y. Tan, and S. Bai. Dragdiffusion: Harnessing diffusion models for interactive point-based image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8839–8849, 2024.

[32] Y. Huang, J. Huang, Y. Liu, M. Yan, J. Lv, J. Liu, W. Xiong, H. Zhang, S. Chen, and L. Cao. Diffusion model-based image editing: A survey. *arXiv preprint arXiv:2402.17525*, 2024.

[33] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[34] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.

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

[36] M. Yang, Y. Du, K. Ghasemipour, J. Tompson, D. Schuurmans, and P. Abbeel. Learning interactive real-world simulators. *arXiv preprint arXiv:2310.06114*, 1(2):6, 2023.

[37] S. Yang, J. Walker, J. Parker-Holder, Y. Du, J. Bruce, A. Barreto, P. Abbeel, and D. Schuurmans. Video as the new language for real-world decision making. *arXiv preprint arXiv:2402.17139*, 2024.

[38] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 conference papers*, pages 1–11, 2024.

[39] C. Doersch, A. Gupta, L. Markeeva, A. Recasens, L. Smaira, Y. Aytar, J. Carreira, A. Zisserman, and Y. Yang. TAP-vid: A benchmark for tracking any point in a video. *Advances in Neural Information Processing Systems*, 35:13610–13626, 2022.

[40] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[41] M. Torne, A. Simeonov, Z. Li, A. Chan, T. Chen, A. Gupta, and P. Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. *arXiv preprint arXiv:2403.03949*, 2024.

[42] T. Samavati and M. Soryani. Deep learning-based 3d reconstruction: a survey. *Artificial Intelligence Review*, 56(9):9175–9219, 2023.

[43] L. Zhou, G. Wu, Y. Zuo, X. Chen, and H. Hu. A comprehensive review of vision-based 3d reconstruction methods. *Sensors*, 24(7):2314, 2024.

[44] H. Chang and A. Boularias. Scene-level tracking and reconstruction without object priors. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3785–3792, 2022. URL https://api.semanticscholar.org/CorpusID:251308689.

[45] H. Chang, D. M. Ramesh, S. Geng, Y. Gan, and A. Boularias. Mono-star: Mono-camera scene-level tracking and reconstruction. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 820–826, 2023. URL https://api.semanticscholar.org/CorpusID:256416010.

[46] W. Gao and R. Tedrake. Surfelwarp: Efficient non-volumetric single view dynamic reconstruction, 2019. URL https://arxiv.org/abs/1904.13073.

[47] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000.

[48] 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*, 65(1):99–106, 2021.

[49] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5865–5874, October 2021.

[50] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10318–10327, June 2021.

[51] A. Cao and J. Johnson. Hexplane: A fast representation for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 130–141, 2023.

[52] S. Fridovich-Keil, G. Meanti, F. R. Warburg, B. Recht, and A. Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12479–12488, June 2023.

[53] Z. Li, Y. Sun, Z. Zheng, L. Wang, S. Zhang, and Y. Liu. Animatable and relightable gaussians for high-fidelity human avatar modeling. *arXiv preprint arXiv:2311.16096*, 2023.

[54] Y. Yuan, X. Li, Y. Huang, S. De Mello, K. Nagano, J. Kautz, and U. Iqbal. Gavatar: Animatable 3d gaussian avatars with implicit mesh learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 896–905, 2024.

[55] Z. Qian, S. Wang, M. Mihajlovic, A. Geiger, and S. Tang. 3dgs-avatar: Animatable avatars via deformable 3d gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5020–5030, 2024.

[56] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. Smpl: A skinned multi-person linear model. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 851–866. 2023.

[57] T. Kuai, A. Karthikeyan, Y. Kant, A. Mirzaei, and I. Gilitschenski. Camm: Building category-agnostic and animatable 3d models from monocular videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6587–6597, 2023.

[58] Y. Zheng, X. Chen, Y. Zheng, S. Gu, R. Yang, B. Jin, P. Li, C. Zhong, Z. Wang, L. Liu, et al. Gaussiangrasper: 3d language gaussian splatting for open-vocabulary robotic grasping. *IEEE Robotics and Automation Letters*, 2024.

[59] O. Shorinwa, J. Tucker, A. Smith, A. Swann, T. Chen, R. Firoozi, M. Kennedy III, and M. Schwager. Splat-mover: Multi-stage, open-vocabulary robotic manipulation via editable gaussian splatting. *arXiv preprint arXiv:2405.04378*, 2024.

[60] G. Lu, S. Zhang, Z. Wang, C. Liu, J. Lu, and Y. Tang. Manigaussian: Dynamic gaussian splatting for multi-task robotic manipulation. In *European Conference on Computer Vision*, pages 349–366. Springer, 2024.

[61] J. Abou-Chakra, K. Rana, F. Dayoub, and N. Suenderhauf. Physically embodied gaussian splatting: A realtime correctable world model for robotics. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=AEq0onGrN2.

[62] H. Jiang, H.-Y. Hsu, K. Zhang, H.-N. Yu, S. Wang, and Y. Li. Phystwin: Physics-informed reconstruction and simulation of deformable objects from videos. *arXiv preprint arXiv:2503.17973*, 2025.

[63] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[64] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[65] T. Ren, Q. Jiang, S. Liu, Z. Zeng, W. Liu, H. Gao, H. Huang, Z. Ma, X. Jiang, Y. Chen, Y. Xiong, H. Zhang, F. Li, P. Tang, K. Yu, and L. Zhang. Grounding dino 1.5: Advance the "edge" of open-set object detection, 2024.

[66] Q. Jiang, F. Li, Z. Zeng, T. Ren, S. Liu, and L. Zhang. T-rex2: Towards generic object detection via text-visual prompt synergy, 2024.

[67] N. Renaud, S. Smeets, and L. J. Corbijn van Willenswaard. nanomesh. URL https://github.com/hpgem/nanomesh.

[68] N. Karaev, I. Makarov, J. Wang, N. Neverova, A. Vedaldi, and C. Rupprecht. Cotracker3: Simpler and better point tracking by pseudo-labelling real videos. *arXiv preprint arXiv:2410.11831*, 2024.

A Appendix

A.1 Related Works

Dynamic Reconstruction. Dynamic reconstruction aims at recovering the geometry, appearance and motion of dynamic scenes from visual data [42, 43, 44, 45]. Early approaches, such as DynamicFusion [21] and SurfelWarp [46], rely on 3D representations like the truncated signed distance function (TSDF) or surfel [47], coupled with explicit deformation graphs to model motion. Recent advances have shifted toward using NeRF [48, 16, 49, 50, 51, 52] and 3D Gaussian [2, 3, 4, 5, 6] as 3D representations. Gaussian splatting-based methods, in particular, have gained rising attention due to their ability of high-quality reconstruction and real-time rendering. For example, 4DGaussians [3] and Deformable-GS [8] encode motion implicitly using deformation networks. Shape-of-Motion [1], GaussianFlow [18] and STG [4] employ shallower models, such as polynomials, which require dense per-Gaussian motion parameters. In contrast to these approaches, our method introduces a sparse and explicit graph-based motion representation for Gaussian splatting, which enables the reconstruction of realistic and complex scenes with greater interpretability and direct and explicit control of the animation.

Explicit Motion Representations. Explicit motion representations offer intuitive visualization and straightforward motion manipulation. Gaussian-based avatar modeling methods [53, 54, 55] rely on parametric human templates, such as SMPL [56], but are inherently limited to human subjects and cannot generalize to arbitrary dynamic objects. SC-GS [7] introduces sparse control points, which can be dragged to create new motions. But SC-GS is restricted to simple, synthetic scenes like those in D-NeRF [50]. CAMM [57] employs kinematic chains for motion representation but is limited to meshes instead of 3D Gaussians and requires occlusion-free videos with clean backgrounds. On the other hand, classical animation techniques [9] utilize explicit motion representations such as deformation graphs [10], harmonic coordinates [11], and cage [12]. However, these methods only focus on applying manually designed motions to mesh surfaces and are not designed for reconstructing motion or geometry from unstructured video data. In contrast, our method introduces a graph-based motion representation for Gaussian splatting, where motion is propagated from sparse graph links to individual Gaussians. 3D Gaussians and motion graphs are jointly optimized from videos of realistic and complex scenes.

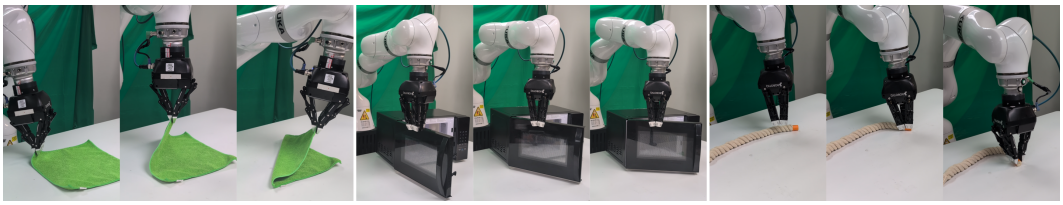


Figure A1: Demonstration of our KUKA robot arm performing cloth folding (left), microwave door adjusting (middle) and rope bending (right), based on planning with our reconstructed motion graphs, without requiring any robot teleoperation data.

Robotics Applications of Dynamic Gaussian Splatting. Recent work has increasingly adopted Gaussian splatting for robotic perception and manipulation. Popular approaches like GaussianGrasper [58] and SplatMover [59] focus on static reconstructions of rigid objects for digital twin applications. For dynamic scenes, ManiGaussian [60] introduces a Gaussian-based world model for manipulation tasks, while EmbodiedGaussians [61] incorporates particle physics for interaction modeling. PhysTwin [62] further advances deformable object reconstruction through mass-spring physics. Despite their impressive results, our work differs in two key aspects: (1) We propose a sparse motion graph representation, while all of these existing work still use dense per-Gaussian motions. (2) Our framework generalizes beyond constrained robotic workspace settings to complex arbitrary scene. We validate our approach on challenging vision benchmarks including iPhone [15] and HyperNeRF [16], demonstrating broader applicability.

A.2 Optimization Details

To reconstruct dynamic scenes from videos, we follow common practices [1, 21] and select the frame where objects occupy the largest image area or the frame with most valid 2D tracks as the canonical frame instead of using the first frame. For simplicity, we still use $t = 0$ to denote canonical frame. We extract instance segmentation masks using Grounding SAM2 [63, 64, 65, 66, 23, 24] and 2D human skeletons using SAPIENS [25]. These annotations are used to initialize the motion graph for each instance at the canonical frame. Specifically, the motion graph is parameterized by $(\theta_t)_{t=0}^{T-1}$ (time-varying parameters) and ϕ (time-independent parameters). We create the graph for each instance at the canonical frame with (θ_0, ϕ) and propagate the parameters across time by setting $\theta_t = \theta_0, \forall t \in [0, T-1]$. We illustrate the motion graph initialization procedure for the canonical frame in Fig. 6 (a). The 3D Gaussians are initialized for every instance and background using procedure from prior work [1]. At each iteration, we transform the 3D Gaussians from the canonical frame $t = 0$ to a target frame t using Eq. 2, and minimize the rendering loss against the corresponding video frame.

Motion Graph Initialization. At $t = 0$, we lift 2D skeletons and instance masks to 3D point clouds using depth images. For kinematic structures such as the human body, we leverage prior knowledge of joint connections to initialize a kinematic tree with standard joint rotations and link lengths. This tree is then fitted to the point cloud skeleton by minimizing the average point-to-link distance via gradient descent. For deformable graphs, we sample points from the instance point cloud using farthest point sampling [22] and connect adjacent points to form the graph.

Instance Level Reconstruction. To reconstruct each instance, we construct a binary matrix $M \in \mathbb{R}^{|\mathcal{G}| \times I}$, where $|\mathcal{G}|$ denotes the number of Gaussians and I denotes the number of instances. M is defined in Eq. 6.

$$M_{ij} = \begin{cases} 1 & i\text{-th Gaussian belongs to instance } j \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The i-th row of M represents the one-hot encoding of the instance index for each Gaussian. We then splat M into a 2D instance mask of dimension $\mathbb{R}^{H \times W \times I}$ and minimize the L1 distance between this rendered mask and the instance mask predicted by Grounding SAM2. In doing so, 3D positions of dynamic Gaussians are grouped to always faithfully represent the geometry of each instance, as illustrated in Fig. 6 (b). To the best of our knowledge, our work is the first to study simultaneous per-instance reconstruction in dynamic Gaussian splatting.

Canonical Frame Regularization. To ensure the learned motion graph remains closely aligned with the object geometry, we minimizes $\sum_{i=1}^{J} \|\mathbf{n}_{i,0} - \hat{\mathbf{n}}_{i,0}\|$ during training, where $\mathbf{n}_{i,0}$ denotes the position of the current i-th joint at the canonical frame, and $\hat{\mathbf{n}}_{i,0}$ represents the position of the same joint but before training. This regularization ensures that the motion graph, which is initialized close to the object geometry prior training, does not drift away from the object during optimization.

2D Keypoints Regularization. We project the 3D joints of the human kinematic tree onto the image plane and minimize the L1 distance between the projected joint positions and the 2D human keypoints predicted by SAPIENS. For our robotic experiments with kinematic trees on non-humanoid objects, see implementation details in Appendix A.3.

A.3 More Details of Real Robot Experiments

We evaluate our method with three real robot manipulation tasks, cloth folding, microwave door adjusting, and rope bending as shown in Fig. 10. We reconstruct the green cloth using a deformable graph, while applying kinematic trees for the microwave and rope. The canonical frame is set to the first frame for all tasks. We use SAM2 for object mask extraction, where we initialize the mask by clicking the object's central region in the first frame, then propagate it across the entire video. We capture the green cloth scene using three D415 RGB-D cameras, while the microwave and rope

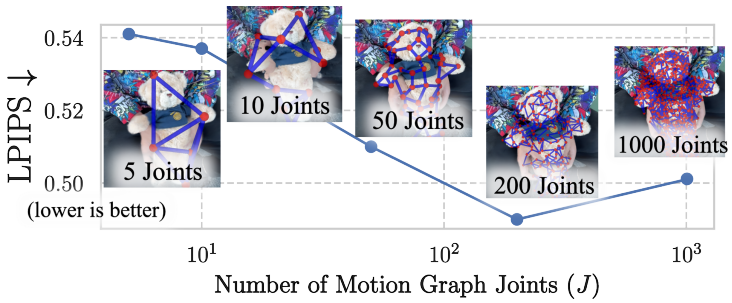


Figure A2: Relationship between novel view rendering quality and motion graph size.

scenes are each recorded with a single D415 RGB-D camera. All the three dynamic objects are reconstructed from human demonstration videos.

For the green cloth, we initialize the motion graph at $t = 0$ using a rectangular mesh generated via NanoMesh [67] (Fig. 10, first column). For the microwave and rope—which lack standardized keypoint structures—we manually define their kinematic trees. The microwave is represented as a 2-link graph, with 3D keypoints annotated in the first ($t = 0$) and final ($t = T - 1$) frames. For the rope, we annotate ten 2D keypoints at the first frame, and track them across frames using CoTracker [68]. The rope's kinematic tree is initialized by projecting the ten manually annotated 2D keypoints at the first frame into 3D space using depth information, with the central keypoint serving as the root. During optimization, we apply 2D keypoint regularization to the rope but omit it for the cloth and microwave.

Given a test scene, we reconstruct its motion graph by optimizing the motion graph parameters of our learned dynamic Gaussian model. The optimization minimizes the L1 distance between the rendered image and the test scene image, while keeping the Gaussian parameters fixed. This process typically converges within a minute, and is akin to pose estimation for rigid objects but uses motion graphs which generalized to more flexible structures.

Given the reconstructed motion graph of a test scene, we simulate object and the corresponding end-effector movements using object-specific geometric heuristics. For cloth, we simulate diagonal folding by sampling random bending axes and angles; for the microwave, we adjust the relative rotation angle between its two rigid links; and for the rope, we bend its single-chain structure at a randomly selected joint with a sampled angle. Despite these simple heuristics, it is possible to integrate the motion graph simulation with a more powerful physics engine, as a future direction that we discussed in Sec. 5. At each timestep, we evaluate candidate motions by comparing their rendered images against the goal image, then select the motion that maximizes the PSNR. For each task, we evaluate performance across ten different goal images, each selected to be achievable through folding or door-adjustment actions. Our method reliably solve the cloth folding and microwave door manipulation tasks (10/10), while rope bending attains a 70% success rate (7/10). We illustrate and discuss a typical failure case in Fig. 12. Fig. A1 demonstrates our Kuka robot performing all three tasks: cloth folding, microwave door adjustment, and rope bending.

A.4 Ablation Studies

Motion Graph Size. Fig. A2 illustrates the relationship between novel view rendering quality and the number of joints in a motion graph. We observe that the best rendering quality for the teddy scene is achieved with a motion graph of 200 joints. However, increasing the number of joints to 1000 leads to a noticeable drop in quality, suggesting the existence of an optimal graph size. This finding highlights the potential of learning adaptive graph structures that dynamically adjust to different objects.

2D Keypoints Regularization. Fig. A3 compares rendered images and motion graphs with and without 2D human keypoint regularization. We observe that enforcing consistency between the

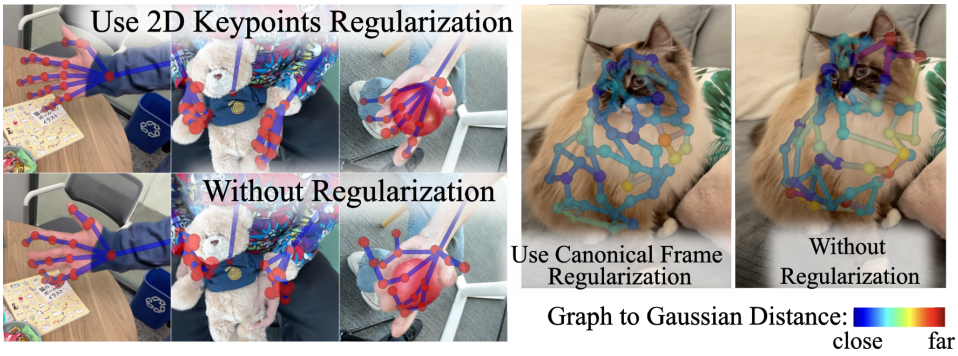


Figure A3: Visualization of rendered images and motion graphs with and without 2D human keypoint regularization (left) and canonical frame regularization (right).

projected 3D kinematic tree and 2D human keypoints (detected by SAPIENS [25]) improves performance, reducing LPIPS by 0.01 to 0.02. More importantly, this regularization produces cleaner and more meaningful motion graphs that align well with the human 3D structure. Without 2D keypoint regularization, the motion graph tends to drift or produce suboptimal structures. For example, if two fingers move together, the kinematic tree may incorrectly control both with a single finger link, leaving other finger links static. This highlights the importance of 2D keypoint regularization in learning accurate and interpretable 3D motion graphs.

Canonical Frame Regularization. The right side of Fig. A3 compares rendered images and motion graphs with and without canonical frame regularization, which minimizes the distance between the learned motion graph's joint positions and their initial positions at the canonical frame. The joints and links of the motion graph are color-coded based on their distance to the Gaussians on the object surface. With canonical frame regularization, the motion graph remains closely aligned with the object surface. Without it, however, the graph joints become spiky (indicated by red-colored joints) and drift outward, detaching from the object.

Figure A4: Visualization of novel view rendering on iPhone dataset compared with other methods. Regions in green are excluded from evaluation, as these pixels are never seen in training videos. LPIPS on bottom left with best colored red. Our method renders sharper, more complete, and perceptually higher-quality novel views.

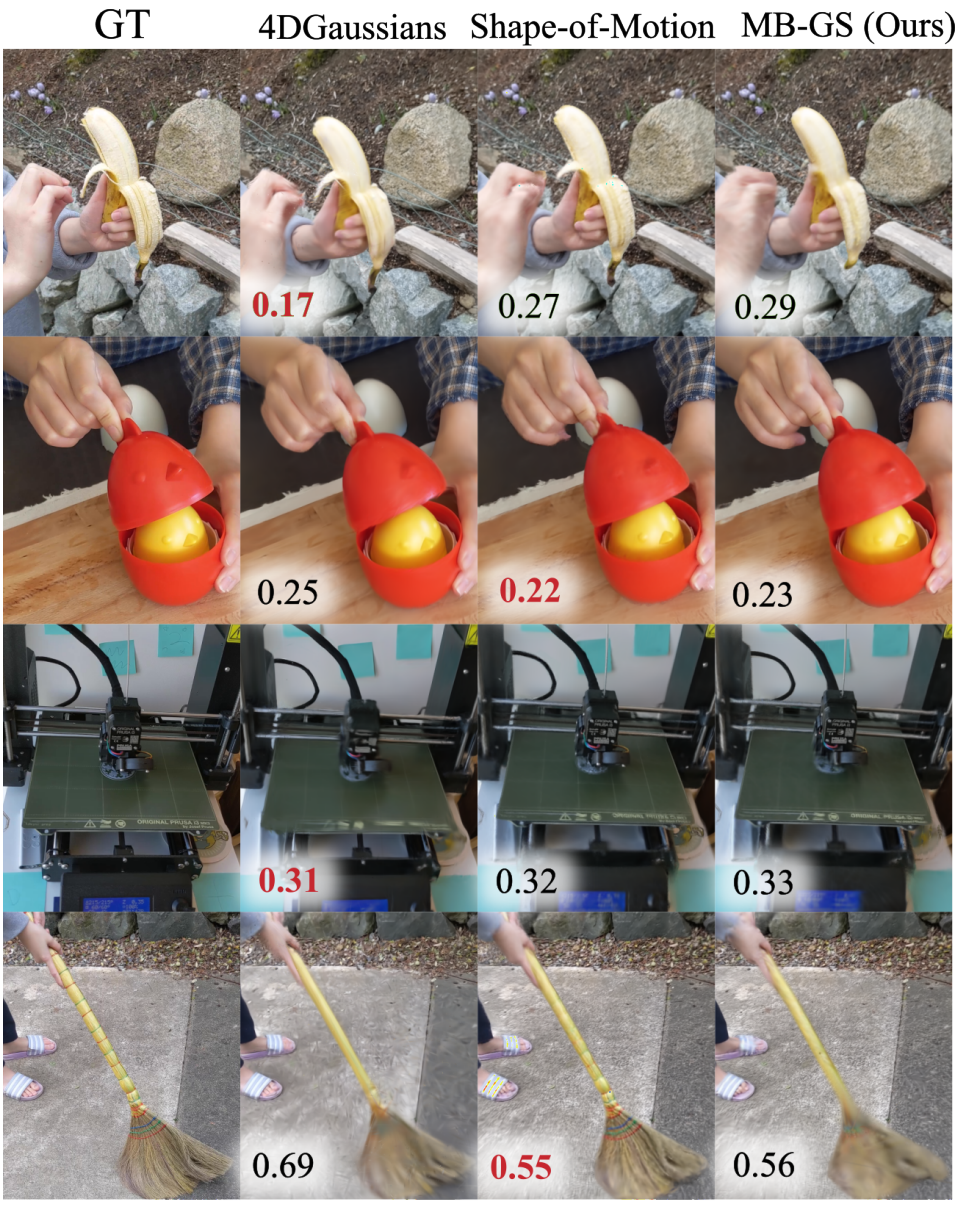


Figure A5: Visualization of results in HyperNerf dataset compared with other methods. LPIPS on bottom left with best colored red. Our method achieves comparable rendering quality and LPIPS with state-of-the-art on the chicken, 3D printer, and broom scenes (rows 2 to 4). On the peel-banana scene, our method produces lower-quality results.