HEIR: Learning Graph-Based Motion Hierarchies Supplemental Material

Cheng Zheng[1*] **William Koch**[1*] **Baiang Li**[1] **Felix Heide**[1,2]
[1]Princeton University [2]Torc Robotics
chengzh, william.koch, baiang.li, fheide@princeton.edu

In this supplementary document, we provide additional details and further results in support of the findings from the main manuscript. The document is structured as follows.

Contents

1 Additional Details on 1D Hierarchical Motion Learning

This section provides further details and justifications for the 1D synthetic experiment described in Section 4.1 of the main paper. The goal of this setup is to test whether our method can identify a motion hierarchy from noisy 1D trajectories. Despite the simplicity of the domain, we show that finding a valid solution is hard, and we provide additional details on the experiment and how we validate hierarchies below.

[*]These authors contributed equally to this work. Listing order is random.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

In a supplementary video, we illustrate the training process: over time, the model learns to decompose the individual node velocities and construct parent-child relationships. Our method reconstructs a valid hierarchy in 73% of runs (Table 1, main paper).

1.1 Difficulty of hierarchical motion learning in the 1D synthetic dataset

Even for only $n = 11$ points the search space for hierarchy matrices is enormous. Because each of the ten non-root nodes can choose any of ten valid parents independently, there are 10^{10} distinct hierarchy matrices that can be sampled. By construction of our synthetic dataset, a valid hierarchy must respect two fixed motion groups $G_1, G_2 \subset \{1, \dots, 10\}$ with $|G_1| = |G_2| = 5$. Choose one distinguished node $p_i \in G_i$ to serve as the parent for its group; every other node $v \in G_i \setminus \{p_i\}$ must then select p_i as its parent. This gives 5 independent choices for p_1 and 5 for p_2. The two parents are related by a single directed edge deciding the global ordering of the groups: either $p_1 \to p_2$ or $p_2 \to p_1$. Hence the number of valid hierarchies is $5 \times 5 \times 2 = 50$ in the entire space. A random hierarchy is therefore correct with probability $50/10^{10} = 5 \times 10^{-9}$, or $5{\times}10^{-7}\%$. Our model attains 73% accuracy on this task, over eight orders of magnitude above chance, demonstrating that it can reliably disentangle inherited motions and recover the hierarchy, even with noisy observations.

1.2 Experimental Details

We train the toy-experiment model with a single message passing layer with learnable edge weights. The absolute velocities are the node features, which are updated through mean-aggregated message passing. We use the Adam optimizer and an initial learning rate of 0.1, combined with a plateau scheduler (decay factor 0.8, patience 20 and a minimum learning rate of 0.02).

To ensure differentiability during sampling, we apply the Gumbel-Softmax trick [3], which is parametrized by a temperature parameter τ_g, which is annealed linearly from 1.5 to 0.3 (with a break-point with $\tau = 0.7$ at epoch 200).

Additionally, we use a custom softmax σ_{τ_s} in the message passing layer to guarantee that every edge retains a strictly positive weight above a threshold, controlled by τ_s, to prevent pruning of edges due to vanishing probabilities with overconfident logits:

$$\sigma_{\tau_s} : \mathbb{R}^d \longrightarrow \mathbb{R}^d, \quad \mathbf{x} \longmapsto \frac{1 - \tau_s}{d} + \tau_s \, \sigma(\mathbf{x}). \tag{1}$$

A value of $\tau_s = 0$ corresponds to a uniform distribution, whereas a value of $\tau_s = 1$ corresponds to the standard softmax. We anneal τ_s similar to τ_g, from 0.6 to 1.0 (break-point with temperature of 0.8 at epoch 200).

1.3 Hierarchy Matrix Verification

There are many admissible hierarchy matrices $H \in \{0, 1\}^{n \times n}$. Because siblings can be permuted and leaf groups reordered, exact equality $H = H^\star$ is too strict. We call two hierarchies *equivalent*, written $H \equiv H^\star$, when they induce the same collection of *depth-1 motion groups*:

$$\mathcal{G}(H) = \big\{ \{p\} \cup \{c \mid \text{parent}(c) = p \text{ and } c \text{ is a leaf}\} \big\}_{p \neq r},$$

where r is the root. Thus $H \equiv H^\star$ if and only if $\mathcal{G}(H) = \mathcal{G}(H^\star)$; ordering inside each group and between groups is irrelevant.

Validation algorithm. Algorithm 1 verifies this equivalence. It first extracts all depth-1 groups of the ground-truth tree; it then iteratively prunes the candidate hierarchy, checking that every set of current leaves matches one of those reference groups. A hierarchy passes exactly when the pruning process removes all non-root nodes.

Algorithm 1 Validate a candidate hierarchy H against ground truth $H^\star$

Require: $H^\star$, H (one–hot parent matrices), root index r
$\mathcal{G} \leftarrow$ depth-1 groups of $H^\star$ {parent + leaf children}
parent$[i] \leftarrow \arg\max_j H_{ij}$ for all i
rem $\leftarrow \{0, \ldots, n-1\} \setminus \{r\}$
while rem $\neq \emptyset$ **do**
  leaves $\leftarrow \{i \in$ rem $\mid$ parent$[j] \neq i \; \forall j \in$ rem$\}$
  if leaves $= \emptyset$ **then**
    return 0
  end if{cycle or no leaf}
  for each p that is parent of some $\ell \in$ leaves **do**
    group $\leftarrow \{\ell \in$ leaves $\mid$ parent$[\ell] = p\}$
    if $p \neq r$ **then**
      group $\leftarrow$ group $\cup \{p\}$
    end if
    if group $\notin \mathcal{G}$ **then**
      return 0
    end if
  end for
  rem $\leftarrow$ rem $\setminus$ leaves
end while
return 1

2 Additional Details on Rotational Hierarchy Learning

2.1 Encoder Architecture

As described in Section 3.4 of the main paper, the encoder $\mathcal{G}(\cdot)$ for rotational hierarchy learning differs only from the one used for the 1D toy example in that it predicts the motion in polar coordinates for each candidate parent-child pair (i, j). As before, the encoder consists of a graph attention network with learnable edge weights. For each edge (i, j) in the fully-connected graph (or k-NN proximity graph, if the size becomes intractable), we maintain learnable attention parameters $\mathbf{w}_{ij} \in \mathbb{R}$ and compute temperatured attention scores with the custom softmax, exactly as in the 1D case.

Given node positions at time t and $t+1$, denoted $\mathbf{x}_i^t$ and $\mathbf{x}_i^{t+1}$, we compute the relative position vectors $\mathbf{r}_{ij}^t = \mathbf{x}_i^t - \mathbf{x}_j^t$ and $\mathbf{r}_{ij}^{t+1} = \mathbf{x}_i^{t+1} - \mathbf{x}_j^{t+1}$ between potential child i and parent j. Let $\mathcal{N}(i)$ denote the set of candidate parents for node i. The aggregated predictions for each node i, i.e., relative velocity in polar coordinates, are computed as weighted sums of the edge-level components. The radial component $\dot{r}_{ij}^t$ measures the rate of change in distance between nodes, computed as:

$$\widehat{\dot{r}}_i^t = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \dot{r}_{ij}^t \quad \text{with} \quad \dot{r}_{ij}^t = \frac{1}{\Delta t}(|\mathbf{r}_{ij}^{t+\Delta t}| - |\mathbf{r}_{ij}^t|) \tag{2}$$

The angular component $\dot{\theta}_{ij}^t$ captures rotational motion around the parent. For 2D motion, this is computed using the change in angle:

$$\widehat{\dot{\theta}}_i^t = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \dot{\theta}_{ij}^t \quad \text{with} \quad \dot{\theta}_{ij}^t = \frac{1}{\Delta t} \text{atan2}\left(\mathbf{r}_{ij}^t \times \mathbf{r}_{ij}^{t+1}, \mathbf{r}_{ij}^t \cdot \mathbf{r}_{ij}^{t+1}\right) \tag{3}$$

.

Note that in practice, we consider $\Delta t = 1$. We also keep the prediction of the relative velocity in Cartesian coordinates, $\widehat{\boldsymbol{\delta}}_i^t$, to provide additional regularization.

2.2 Experimental Details

Dataset. For the rotational experiment, we construct a synthetic planetary system dataset with $T = 100$ timesteps. The system contains eleven bodies: one central star, four planets, and six moons. Each planet orbits the star with a fixed radius and angular velocity, while moons orbit their respective planets following circular trajectories. Ground-truth comparisons are simple, as there only exists a

Table 1: **Parameters of the synthetic planetary system dataset.** Each planet orbits the central star, and each moon orbits its respective planet. The dataset provides ground-truth hierarchical relations for the rotational experiment.

Body	**Parent**	**Radius** r_i	**Angular Velocity** ω_i	**Phase** ϕ_i **(rad)**
Star (root)	Star	0.0	0.0	0.00
Planet 1	Star	1.0	4.0	0.00
Planet 2	Star	1.8	3.0	1.57
Planet 3	Star	2.5	2.5	3.14
Planet 4	Star	3.2	2.0	4.71
Moon 1 of Planet 3	Planet 3	0.6	12.5	0.00
Moon 1 of Planet 4	Planet 4	0.5	15.0	0.00
Moon 2 of Planet 4	Planet 4	0.7	14.0	3.14
Planet 5 (outer)	Star	4.5	1.25	0.00
Moon 1 of Planet 5	Planet 5	1.0	7.5	0.00
Moon 2 of Planet 5	Planet 5	1.3	6.0	2.09

single valid hierarchy matrix for this experiment. We highlight that this dataset does not aim to be physically accurate.

We construct the dataset both with and without noise. For the version with noise, Gaussian noise with standard deviation $\sigma = 0.05$ is added to the node positions at every timestep, to simulate measurement uncertainty. We also provide ablation experiments on additional noise levels.

3 Additional Details of 3D Scene Deformation

3.1 Encoder Architecture

Model Structure. The encoder $\mathcal{G}(\cdot)$ in our hierarchical motion model consists of two graph attention layers designed to progressively infer motion hierarchies at different levels of abstraction. The first graph attention layer $\mathcal{G}_1(\cdot)$ takes every individual motion element as input and computes local attention scores over candidate parent-child relationships. These attention scores remain consistent across all time steps within each scene. From the first-layer attention scores, we sample an initial hierarchy matrix, which is then converted into a directed graph using NetworkX's DiGraph structure. We identify weakly connected components in this graph, each typically containing cycles, as distinct motion groups.

To learn group-level relationships, we select a representative node from each motion group. Specifically, the representative is chosen as the node with the largest number of descendants, corresponding naturally to the most parent-like node within that group. Anchored by these representative nodes, we construct a fully-connected group-level attention graph in the second graph attention layer $\mathcal{G}_2(\cdot)$.

Graph Attention Layer. Each graph attention layer in our model is implemented using a single 4-head `GATConv` module from PyTorch Geometric, with `in_channels` and `out_channels` of 3. It performs multi-head attention over the input graph without self-loops and dropouts, and the output node features and the edge attention weights are averaged across all attention heads. It ensures that each node aggregates information stably from all its neighbors to produce a single unified representation. We apply this same GATConv architecture in both $\mathcal{G}_1(\cdot)$ and $\mathcal{G}_2(\cdot)$, differing only in their input features and graph connectivity.

3.2 Experimental Details

Datasets. We extract the motion data X^t for each scene in D-NeRF dataset [6] using the training pipeline introduced in [2], where each scene contains approximately 100k Gaussians. Constructing a hierarchy matrix for this scale of Gaussians demands significant computational resources. To address this, we train a low-resolution representation by limiting the total number of Gaussians to approximately 5k per scene, following the same training procedure. We equally divide the full motions in the dataset into $T = 50$ time steps to obtain the discrete positions. During training, we learn the hierarchy matrix on the downsampled representation. At inference time, we use the learned

hierarchy to apply deformations to the low-resolution Gaussians and use weight-based skinning to transfer the deformation to the high-resolution Gaussians, which are then used to render the final high-quality deformed scene.

Training. Learning directed acyclic graph (DAG) representations is challenging when the number of vertices N increases. The number of potential edges grows quadratically with N, raising the risk of cycles forming during graph structure discovery. Since Gumbel-Softmax does not inherently enforce acyclicity, costly cycle detection or regularization steps must be integrated into training, complicating optimization and scalability as graphs become larger. Here, we implement a two-stage optimization strategy to progressively learn the corresponding hierarchy matrix.

In *Stage 1*, we only optimize the parameters in $\mathcal{G}_1(\cdot)$, where the edges are computed over k-nearest neighbors with $k = 100$. A batch size of 1 is used, and data from different time steps is stacked along the batch dimension to enable consistent graph processing. Their attention scores are shared across all time steps within each scene and used to sample a hierarchy matrix via Gumbel-Softmax, with a temperature $\tau = 1.6$. Hierarchies are sampled with $S = 1$ sample per pass. Motion is then reconstructed through propagation over the sampled hierarchy, and we supervise the predicted motion from $\mathcal{G}_1(\cdot)$ with an L1 velocity loss. The model is trained for 50 epochs using the Adam optimizer with a learning rate of 0.002. A `ReduceLROnPlateau` scheduler is used to reduce the learning rate by a factor of 0.8 if loss stagnates for 10 epochs.

In *Stage 2*, we freeze the parameters in $\mathcal{G}_1(\cdot)$ and introduce a second graph attention layer $\mathcal{G}_2(\cdot)$ to learn higher-level, group-wise motion structures. To ensure the separation of hierarchical levels, we remove all incoming edges to the representative nodes in each group, so they do not receive messages from neighbors in the first layer and instead aggregate information solely through the second layer. The model is trained for 100 epochs using the same settings and hyperparameters. The outputs from both layers are combined, according to their respective learned hierarchies, to produce the final motion prediction. In addition to the loss in *Stage 1*, we add a distance correlation loss with weights of $\lambda_d = 0.1$, which encourages the learned group-level attention weights to be inversely correlated with the spatial distances between groups. It is based on the Pearson correlation between the attention weights and the pairwise Euclidean distances of group representatives:

$$\mathcal{L}_{\text{d}} = \lambda_d \cdot (r+1)^2, \; r = \frac{\sum_{i,j} \bar{d}_{ij} \cdot \bar{w}_{ij}}{\sqrt{\sum_{i,j} \bar{d}_{ij}^2} \cdot \sqrt{\sum_{i,j} \bar{w}_{ij}^2} + \epsilon} \tag{4}$$

where $\bar{d}_{ij} = \tilde{d}_{ij} - \text{mean}(\tilde{d})$, $\bar{w}_{ij} = w_{ij} - \text{mean}(w)$, and w_{ij}, $\bar{d}_{ij}$ are the mean attention weight and the normalized Euclidean distance from representative nodes in group j to group i. Since the Pearson correlation coefficient r lies in the range $[-1, 1]$, we shifted and squared it into a positive scalar loss.

We train our model and conduct all the experiments on an NVIDIA RTX 3090 GPU; the training on each scene takes approximately 40 minutes.

3.3 Scene Deformation Details

Keypoint Selection. The deformation process begins with the user specifying a keypoint—typically a parent node in the hierarchy—as the anchor for deformation. We then construct a deformation group by automatically traversing the learned hierarchy to collect all descendants of the keypoint, ensuring that the affected region reflects the underlying motion structure. Optionally, we select a sparse set of 512 Gaussians from the full set using the Farthest Point Sampling (FPS) algorithm and define the deformation group as the intersection between this sparse subset and the keypoint's descendants. This strategy reduces computational overhead and enables smooth, real-time deformation rendering.

Deformation Computation. We apply the displacements of Gaussians within the deformation group by assuming they form a rigid object. Given the deformation of the keypoint represented by a translation vector T_k and a rotation matrix R_k, the updated position of the Gaussian i inside the deformation group is calculated with $p_i' = R_k(p_i - p_k) + p_i + T_k$, where p_k and p_i are the initial position of the keypoint and the Gaussian i. For the rest of the Gaussians outside the deformation group, we perform global refinement using As-Rigid-As-Possible (ARAP) optimization over all Gaussians. The ARAP algorithm takes the directly manipulated handles as positional constraints and computes new positions for the remaining Gaussians that best preserve local rigidity. Note that these operations are all done on the downsampled Gaussians.

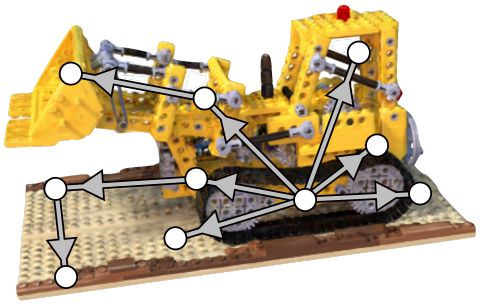

Figure 1: **Learned hierarchy in the Excavator scene**. We overlay the abstracted learned hierarchy on the excavator scene. The center of the excavator body is recognized as the root of motion, and the shovel is correctly assigned as the child of the upper-left part of the excavator body.

Weights-based Skinning. To propagate deformations to the full Gaussian set, we compute skinning weights using a distance-based soft assignment. For each Gaussian, we identify its $k = 16$ nearest points and extract the squared distances. We apply a softmax over the negative distances to compute the skinning weights. This ensures that nearby points exert greater influence, and that all weights for a given Gaussian sum to one, enabling smooth and spatially consistent deformation transfer.

3.4 Additional Evaluations

We evaluated our method on the full D-NeRF dataset, besides the scenes shown in the main paper. The reconstruction component of SC-GS that we use as input for our method achieves PSNR 40.72, SSIM 0.998, and LPIPS 0.021 before performing any scene deformation with our test-time method. The average metrics of our method versus SC-GS after deformation over the full D-NeRF dataset are PSNR(↑) 20.35/19.30, SSIM(↑) 0.9438/0.9366, CLIP-I(↑) 0.9635/0.9621 and LPIPS(↓) 0.0453/0.0659. For additional reference-free evaluation, we provide FID (Frechet Inception Distance) scores of 68.65 (our method) v.s. 107.03 (SC-GS). Note that this number is relatively high compared to values reported in generative models as the D-NeRF subset contains few images.

We also visualize the learned hierarchy on top of the rendered Gaussian scene to see if relative motions are correctly observed. Note that the Gaussian scenes don't have a ground-truth hierarchy structure as in the toy examples; we use the figure here as a sanity check. To ensure that meaningful hierarchies are clearly presented, we cluster the nodes into 10 macro nodes, whose locations are the centroids of the clusters. We build the hierarchy between the macro nodes using their relative depths (distance to the root node), where the depth of a macro node is the average depth of all nodes in the cluster. We overlay the abstracted learned hierarchy on the scene, with arrows pointing from parent to child. We can easily observe the center of the excavator body being recognized as the root, and the shovel being the child of the upper left part of the excavator body.

4 Ablation Experiments

Fraction of Time Steps. We evaluated how the number of available time steps in the training data influences the accuracy of the hierarchy reconstruction. In the 1D synthetic experiment, we use a fraction of the time steps instead of the full time steps ($T = 200$) for training and measure hierarchy reconstruction accuracy over 100 iterations. As shown in Fig. 2, accuracy climbs steadily as the fraction of training data increases, reaching roughly 45% accuracy when half the frames are available, and begins to plateau once approximately 70% of the time frames are included. It aligns well with the expectation of this data-driven method, where enough observation of meaningful motions is necessary to learn the underlying hierarchy.

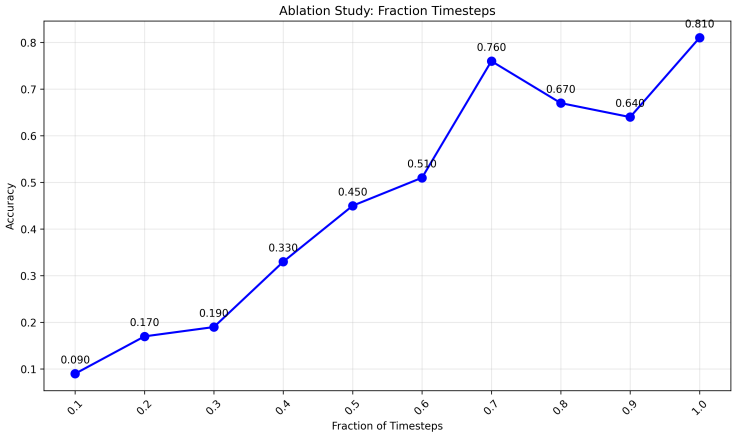

Figure 2: **Ablation study on fraction of time steps**. We evaluated the effects of the number of time steps in the training data on 1D hierarchy reconstruction. The plot shows that the accuracy almost linearly increases with the growth of available time steps, and reaches a plateau of $\sim 70\%$ accuracy when 70% time steps are available.

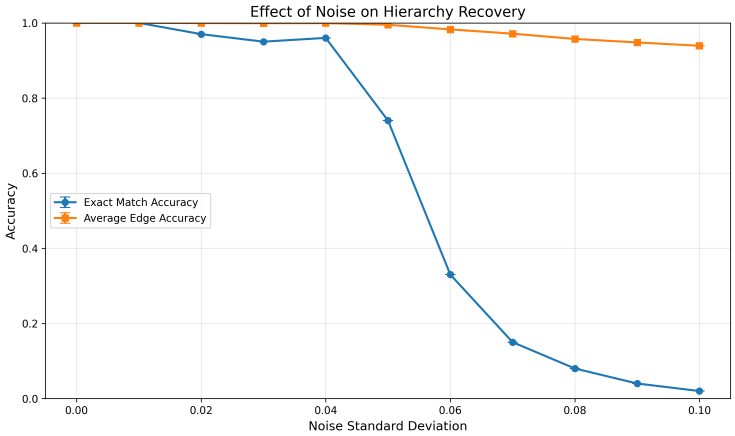

Figure 3: **Ablation study on impact of noise in observed data**. The model achieves 100% hierarchy match accuracy (blue) if the amount of noise is very limited or zero, and drops quickly for larger noise levels. The average edge accuracy (orange), however, remains stable - indicating that the reconstruction fails only due to a few incorrect edges.

Impact of Noise in Training Data. We evaluate the robustness of hierarchy recovery for the rotational reconstruction under varying levels of Gaussian noise applied to node positions, with standard deviations $\sigma \in [0.0, 0.1]$. Each configuration is trained for 500 epochs and averaged over 100 independent runs, using the same hyperparameters as in the main experiment (i.e., $\lambda_{\dot{r}} = 12.0$, $\lambda_{\dot{\theta}} = 0.0$, $\lambda_{\delta} = 0.8$ and $\lambda_{\Lambda} = 6.0$). As shown in Fig. 3, exact match accuracy (blue) measures whether the entire hierarchy is recovered correctly, while average edge accuracy (orange) measures the fraction of correctly recovered parent–child edges, independent of whether the global structure is fully correct. The model maintains nearly perfect edge-level accuracy up to moderate noise levels, whereas exact match accuracy drops sharply beyond $\sigma \approx 0.05$. This behavior likely arises as noise begins to dominate the small orbital displacements of planets and moons closer to the star, while nodes with larger radii remain less affected. Most relations therefore continue to be correctly predicted, despite the overall reconstruction being incorrect. We note that computing the average edge accuracy, which we introduce here, is not possible in the 1D toy example, as the 1D toy example admits many different valid hierarchy reconstructions.

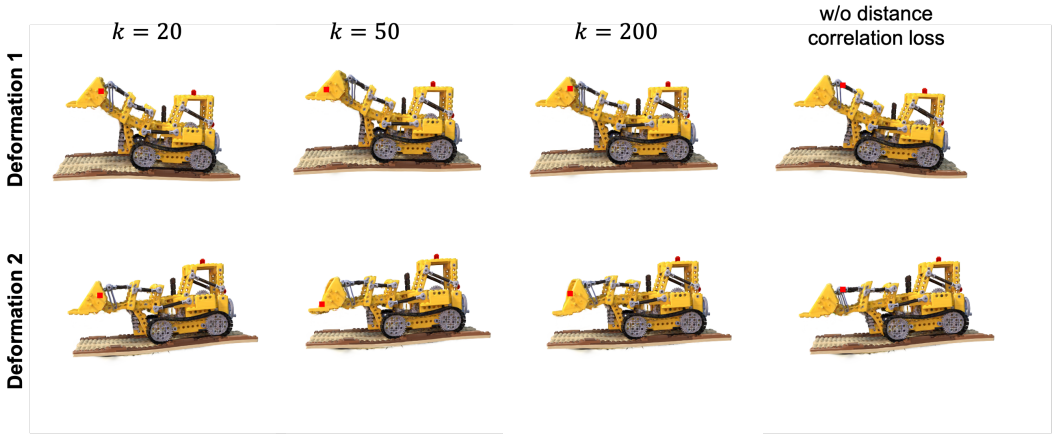

Figure 4: **Ablation study of GS experiments**. We evaluate the effects of varying the number of neighbors k in the proximity graph and the distance correlation loss in hierarchy learning. We apply the same deform operations on the scene as in the main paper. The left three columns show results under k-NN values of 20, 50, and 200. Smaller k leads to rigid, less expressive deformation, while larger k increases flexibility but introduces instability. The rightmost column shows results without the distance correlation loss, which leads to noticeable artifacts in the ground and shovel during deformation.

Proximity Graph Sparsity. In the initial proximity graph, edges are determined using k-nearest neighbors in Euclidean space. The choice of k directly affects the sparsity of the graph and thus the searching space for the learned hierarchy. A smaller k enforces stricter locality, potentially missing long-range dependencies, while a larger k allows for more flexible hierarchy formation but may introduce noise and increase computational overhead. To study the impact of this parameter, we conduct an ablation study by varying the value of k and evaluating the resulting deformation quality. As shown in Fig.4, all settings $(k = 20, 50, 200)$ preserve basic structural coherence. However, we observe that $k = 20$ results in more rigid local behavior, making deformation harder to propagate, whereas $k = 200$ offers increased flexibility at the cost of reduced stability. Moderate values such as $k = 50 \sim 100$ provide a good balance between locality and global consistency.

Distance Correlation Loss. In optimization *Stage 2*, we apply a distance correlation loss to encourage group-level attention to align with spatial proximity, promoting stronger interactions between nearby motion groups. Without this loss, group attention becomes less spatially coherent, leading to a slight drop in deformation quality in Fig. 4. We can observe distortions of the ground and the shovel compared to the scenario with distance correlation loss. By incorporating this loss, spatial consistency among the motion groups is maintained, which also improves structural coherence during deformation.

5 Additional Comparison

Most existing 3D methods do not support user-interactive scene deformation, with the exception of SC-GS [2] (included in the main paper) and 3DGS-Drag [1], whose source code was unavailable at the time of this study. For completeness, we additionally compare our method with PDS [4], a 3D editing approach that aligns the stochastic latent spaces of source and target by sampling in the target latent space to match distributions. We also include SDE-Drag [5], a 2D editing method that formulates point-based content dragging as a stochastic differential equation (SDE) process.

For SDE-Drag, we enable the *train_lora* option as recommended in the source code to ensure optimal performance. Fig. 5 shows SDE-Drag either barely deforms the scene or introduces severe distortion under the same deformation operation indicated by the arrows on the source image. For PDS, we fine-tune the model on the D-NeRF dataset using the pretrained Stable Diffusion v1.5, following the official implementation. Fig. 6 shows that PDS fails to generate meaningful results on the same input scenes. We attribute the failure of both methods to a domain gap: they are fine-tuned on Stable

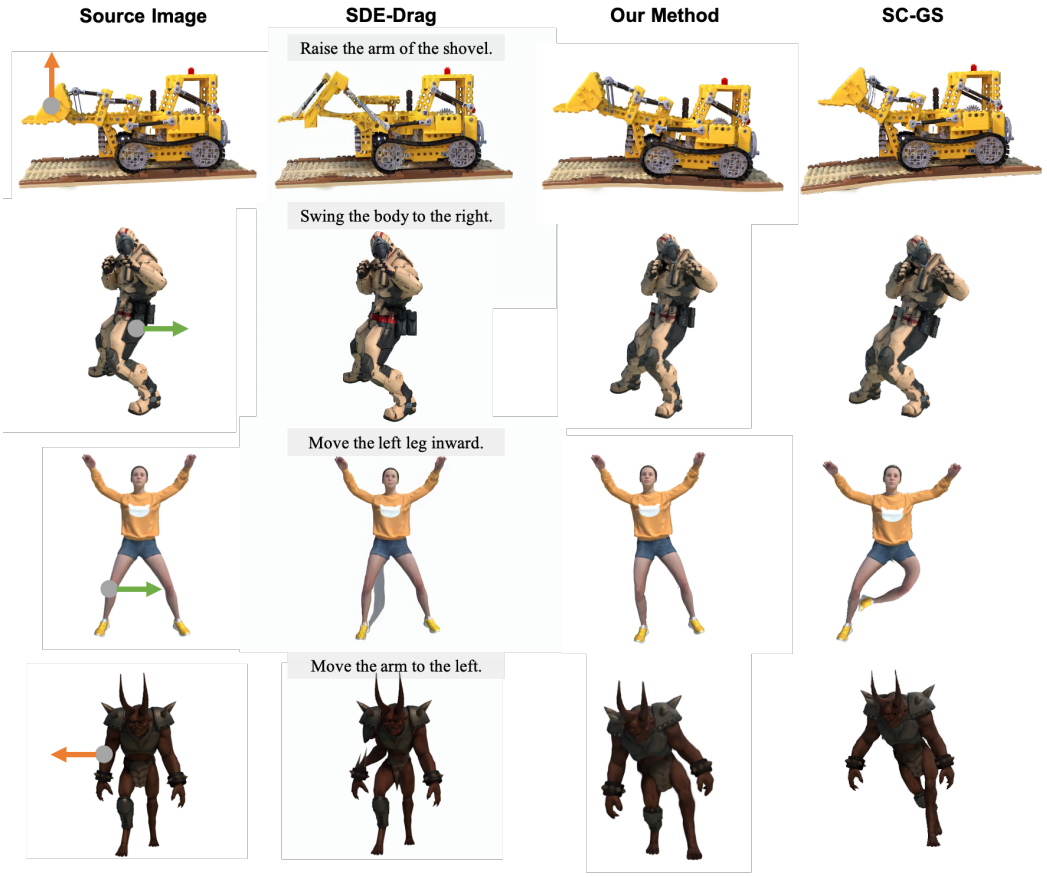

Figure 5: Scene editing results using SDE-Drag [5]. In the generated editing images using SDE-Drag, either no clear deformation or severe distortion can be observed compared to our method on the right. It is also worse than the SC-GS method we compared in the main text. The drag points and prompts we used to generate the edited images are labeled on the images. Note that SDE-Drag edits on a 2D source image, while our method and SC-GS deform a 3D scene represented by Gaussian splatting.

Diffusion v1.5 [7], which is trained on more complex, natural images and does not align well with the D-NeRF dataset.

Another established field very relevant to our method is the rigging techniques. Although RigNet [9] and UniRig [11] generate a skeleton that has a very similar structure to our hierarchy graph, RigNet's BoneNet and UniRig's Skeleton-Tree GPT are both trained on thousands of meshes with ground-truth joints and skinning weights. Our hierarchy graph is directly learned from the motion from the scene via a self-supervised reconstruction loss; no pre-rigged data is required. BANMo/DreaMo [10, 8] learn where to place deformable handles ('neural bones' in the paper) so the object can be rendered consistently across video frames whereas our method learns how those parts influence each other over time. This fundamental difference in objective makes their skeleton (only available in DreaMo) adjacency-based, and doesn't yield the parent-child relationships we target. In fact, our goal of motion-centric hierarchy is complementary to many rigging techniques. For example, a supervised rigger could seed an initial topology, while our optimization refines parent-child directions using dynamics. It is especially useful for long-range dependencies that K-NN initialization may miss.

References

[1] Jiahua Dong and Yu-Xiong Wang. 3DGS-Drag: Dragging gaussians for intuitive point-based 3D editing. In *Proceedings of the International Conference on Learning Representations*, 2025. 8

[2] Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. SC-GS: Sparse-controlled gaussian splatting for editable dynamic scenes. In *Proceedings of the IEEE/CVF Conference on*

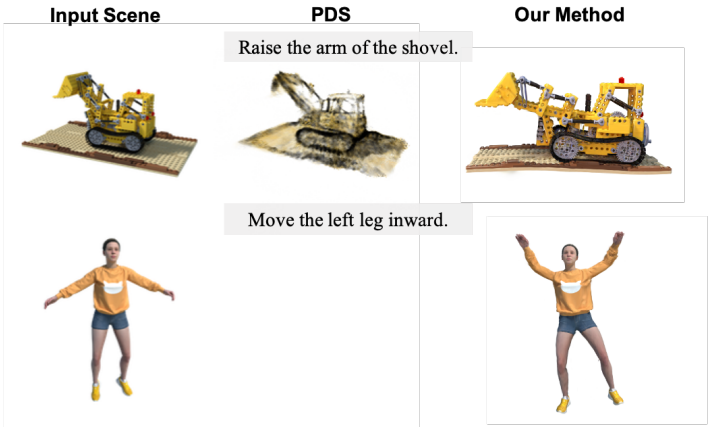

Figure 6: Scene editing results using PDS [4]. We perform fine-tuning on this model under the D-Nerf dataset using the pre-trained stable diffusionv1.5 [7], as described in the paper and the source code. PDS fails in both cases, generating blank or low-quality scenes.

Computer Vision and Pattern Recognition, pages 4220–4230, 2024. 4, 8

[3] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. 2

[4] Juil Koo, Chanho Park, and Minhyuk Sung. Posterior distillation sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13352–13361, 2024. 8, 10

[5] Shen Nie, Hanzhong Allan Guo, Cheng Lu, Yuhao Zhou, Chenyu Zheng, and Chongxuan Li. The blessing of randomness: Sde beats ode in general diffusion-based image editing. *arXiv preprint arXiv:2311.01410*, 2023. 8, 9

[6] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021. 4

[7] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 9, 10

[8] Tao Tu, Ming-Feng Li, Chieh Hubert Lin, Yen-Chi Cheng, Min Sun, and Ming-Hsuan Yang. Dreamo: Articulated 3d reconstruction from a single casual video. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2269–2279. IEEE, 2025. 9

[9] Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. Rignet: Neural rigging for articulated characters. *arXiv preprint arXiv:2005.00559*, 2020. 9

[10] Gengshan Yang, Minh Vo, Natalia Neverova, Deva Ramanan, Andrea Vedaldi, and Hanbyul Joo. Banmo: Building animatable 3d neural models from many casual videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2863–2873, 2022. 9

[11] Jia-Peng Zhang, Cheng-Feng Pu, Meng-Hao Guo, Yan-Pei Cao, and Shi-Min Hu. One model to rig them all: Diverse skeleton rigging with unirig. *ACM Transactions on Graphics (TOG)*, 44(4):1–18, 2025. 9