Speedy Deformable 3D Gaussian Splatting:
Fast Rendering and Compression of Dynamic Scenes

Allen Tu[*] **Haiyang Ying**[*] **Alex Hanson** **Yonghan Lee**

Tom Goldstein **Matthias Zwicker**

University of Maryland, College Park

https://speede3dgs.github.io/

Figure 1: Our SpeeDe3DGS pipeline accelerates rendering speed by $49.36\times$, reduces the number of Gaussians by $12.77\times$, and shortens training time by $4.64\times$ on the HyperNeRF *chicken* scene while producing image quality comparable to the baseline Deformable 3D Gaussian Splatting model.

Abstract

Recent extensions of 3D Gaussian Splatting (3DGS) to dynamic scenes achieve high-quality novel view synthesis by using neural networks to predict the time-varying deformation of each Gaussian. However, performing per-Gaussian neural inference at every frame poses a significant bottleneck, limiting rendering speed and increasing memory and compute requirements. In this paper, we present Speedy Deformable 3D Gaussian Splatting (SpeeDe3DGS), a general pipeline for accelerating the rendering speed of dynamic 3DGS and 4DGS representations by reducing neural inference through two complementary techniques. First, we propose a temporal sensitivity pruning score that identifies and removes Gaussians with low contribution to the dynamic scene reconstruction. We also introduce an annealing smooth pruning mechanism that improves pruning robustness in real-world scenes with imprecise camera poses. Second, we propose GroupFlow, a motion analysis technique that clusters Gaussians by trajectory similarity and predicts a single rigid transformation per group instead of separate deformations for each Gaussian. Together, our techniques accelerate rendering by $10.37\times$, reduce model size by $7.71\times$, and shorten training time by $2.71\times$ on the NeRF-DS dataset. SpeeDe3DGS also improves rendering speed by $4.20\times$ and $58.23\times$ on the D-NeRF and HyperNeRF *vrig* datasets. Our methods are modular and can be integrated into any deformable 3DGS or 4DGS framework.

---

[*]denotes equal contribution

Preprint. Under review.

1 Introduction

Novel view synthesis is a long-standing problem in computer vision that aims to synthesize photorealistic images from novel viewpoints given a limited set of input images. Neural Radiance Fields (NeRFs) [31] have revolutionized this domain by modeling scenes as continuous volumetric functions using multilayer perceptrons (MLPs) optimized via differentiable volume rendering. To model dynamic scenes, NeRF variants incorporate time as an additional dimension, enabling 4D scene reconstruction through techniques such as deformation fields [37]. Despite their high visual fidelity, NeRF-based techniques suffer from slow rendering and high neural inference costs.

3D Gaussian Splatting (3DGS) [21] has emerged as a faster alternative, representing scenes as point clouds of 3D Gaussian primitives rendered via a differentiable rasterizer. While originally developed for static scenes, 3DGS has been extended to dynamic settings through both explicit and implicit motion modeling. For instance, Dy3DGS[29] fits explicit 6-DoF trajectories to each Gaussian, while Deformable 3DGS [50] and 4DGS [45] use deformation networks to predict time-varying geometry. However, these dynamic extensions require per-Gaussian neural inference at each frame, which significantly degrades rendering speed as model size increases.

In this paper, we propose **Speedy Deformable 3D Gaussian Splatting (SpeeDe3DGS)**, a pipeline that significantly accelerates dynamic 3DGS representations by reducing neural inference through two complementary strategies. Our methods are motivated by two key insights. First, since a deformation is predicted for every Gaussian, pruning Gaussians with low contribution to the dynamic scene reconstruction directly lowers the inference load. Second, Gaussians that follow similar motion patterns can be grouped, enabling a single shared trajectory to be predicted for each group instead of separate deformations for each Gaussian. Although we integrate our approach into the Deformable 3D Gaussian Splatting (De3DGS) framework, our methods are modular and can be applied to any similar deformable 3DGS or 4DGS framework. On the real-world NeRF-DS dataset, SpeeDe3DGS accelerates rendering speed by $10.37\times$, compresses model size by $7.71\times$, and shortens training speed by $2.71\times$ – all while maintaining high image quality. We also observe rendering speedups of $4.20\times$ and $58.23\times$, model size reductions of $5.82\times$ and $18.82\times$, and training time decreases of $2.07\times$ and $5.22\times$ on the synthetic D-NeRF [37] and real-world HyperNeRF [35] *vrig* datasets, respectively.

In summary, we propose the following contributions:

1. **Temporal Sensitivity Pruning Score:** A pruning strategy that reduces neural inference by identifying and removing Gaussians with low contribution to the dynamic scene. We also introduce an Annealing Smooth Pruning (ASP) mechanism that improves the robustness of pruning to imprecise camera poses in real-world scenes.

2. **GroupFlow:** A motion analysis method that clusters Gaussians based on trajectory similarity and predicts a shared rigid motion for each group, replacing per-Gaussian deformation and thereby reducing neural inference.

2 Related Works

Neural Radiance Fields (NeRF) [31] have demonstrated impressive performance in photorealistic novel view synthesis by modeling scenes as continuous volumetric functions using multilayer perceptrons (MLPs) optimized via differentiable volume rendering. To extend NeRF to dynamic scenes, many methods incorporate time as an additional dimension to enable 4D novel view synthesis. Some approaches augment NeRF with explicit temporal modeling [5, 11, 42], while others adopt a canonical space and learn deformation fields to capture motion [34, 35, 37]. More recent approaches improve efficiency by employing grid-based representations to accelerate training and inference without sacrificing visual quality [4, 9, 10, 12, 14, 26, 30, 36, 43].

Despite their visual fidelity, NeRF-based methods suffer from slow rendering and rigid scene representation. To overcome these limitations, 3D Gaussian Splatting (3DGS) [21] was introduced as an efficient alternative that models scenes as point clouds of 3D Gaussians rendered via differentiable rasterization. Although originally designed for static scenes, a growing number of methods extend 3DGS to the temporal domain for dynamic scene reconstruction. Dy3DGS [29] fits an explicit 6-DoF trajectory to each Gaussian and applies physical regularization to for temporal consistency. Deformable 3D Gaussian Splatting (De3DGS) [50] and 4D Gaussian Splatting (4DGS, CVPR) [45]

| Baseline | Pruning | GroupFlow | Pruning + GroupFlow |

Figure 2: **Visual comparison of our baseline and SpeeDe3DGS methods.** The combination of pruning and GroupFlow delivers vastly faster results. Top: *basin* from the NeRF-DS dataset. Middle: *trex* from D-NeRF. Bottom: *vrig-chicken* from HyperNeRF. Annealing smooth pruning, discussed in Section 4.1.2, is used in the real-world scenes.

use implicit deformation fields to model motion, with the latter employing HexPlane [4] for efficient encoding. 4DGS (ICLR) [49] jointly embeds space and time in a full 4D representation, enabling fast rendering but requiring significantly more primitives.

2.1 Gaussian Pruning

Many Gaussians in static 3DGS reconstructions are redundant, and pruning them can significantly reduce model size and improve rendering speed [7, 8, 16, 17, 32]. Most pruning approaches learn a per-Gaussian pruning indicator [23, 53] or compute a heuristics-based score to rank remove Gaussians with the lowest contribution to the scene [1, 2, 6, 8, 13, 24, 28, 32, 33]. Several methods extend these ideas to dynamic scenes by adapting pruning scores to account for temporal variation [19, 25, 52].

PUP 3D-GS [17] introduces a Hessian-based sensitivity score that quantifies the contribution of each Gaussian to scene reconstruction, enabling substantially more Gaussians to be pruned than heuristics-based methods while preserving visual fidelity. Speedy-Splat [16] reduces the memory requirement of this score and integrates it into the 3DGS training pipeline, achieving a $6.71\times$ rendering speed-up and a $10.6\times$ reduction in model size. In Section 4.1, we extend Speedy-Splat's sensitivity score to the temporal domain and propose a noise-based regularization mechanism designed to improve pruning robustness in real-world scenes.

2.2 Motion Analysis

Another strategy for compressing dynamic scenes is to reduce temporal complexity by decomposing the scene into static and dynamic regions. Since most areas in real-world scenes remain static, several methods identify dynamic regions through motion analysis [38, 44, 46, 47, 54] or use self-supervised or segmentation-based techniques to assign Gaussians accordingly [44, 47]. By restricting 4D motion modeling to dynamic regions, these approaches substantially reduce computational overhead.

However, static and dynamic decomposition is less effective for compressing objects with complex non-rigid motion. To address this, several works have explored group-based motion modeling. SC-GS [18] and SPGaussian [41] use control-point-based linear blend skinning with MLPs to model motion, while RigGS [51] extracts sparse skeletons but still requires a residual MLP to preserve quality. DynMF [22] uses optimizable polynomial motion bases, though inference is still costly due

to the overhead of basis weights and neural computation. In contrast, our proposed approach in Section 4.2 fits per-Gaussian trajectories and then explicitly groups Gaussians by motion similarity, allowing a single rigid motion to be optimized per group and substantially reducing inference cost.

3 Background

3.1 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) [21] represents scenes as parametric, point-based models composed of 3D Gaussians. Given a set of ground truth training images $\mathcal{I}_{gt} = \{\boldsymbol{I}_i \in \mathbb{R}^{H \times W}\}_{i=1}^{M}$, the scene is initialized using Structure from Motion (SfM) to produce image pose estimates and a sparse point cloud that serves as the initial means for the 3D Gaussians. Each image is paired with its corresponding pose in $\mathcal{P}_{gt} = \{\phi_i \in \mathbb{R}^{3 \times 4}\}_{i=1}^{M}$ and used to optimize the scene.

Each 3D Gaussian primitive $\mathcal{G}_i$ is parameterized by three geometry parameters – mean $\mu_i \in \mathbb{R}^3$, scale $s_i \in \mathbb{R}^3$, and rotation $r_i \in \mathbb{R}^4$ – and two appearance parameters – view-dependent spherical harmonic color $h_i \in \mathbb{R}^{16 \times 3}$ and opacity $\sigma_i \in \mathbb{R}$. The set of all parameters can be described as:

$$\mathcal{G} = \{\mathcal{G}_i = \{\mu_i, s_i, r_i, h_i, \sigma_i\}\}_{i=1}^{N}, \tag{1}$$

where N is the number of Gaussians in the model. Given camera pose ϕ, the scene is rendered by projecting all Gaussians to image space and compositing them via alpha blending. The value of the 2D projection of Gaussian $\mathcal{G}_i$ at pixel p is given by g_i:

$$g_i = e^q, q = -\frac{1}{2}(p - \mu_{i_{2D}})\boldsymbol{\Sigma_{i_{2D}}}^{-1}(p - \mu_{i_{2D}})^T, \tag{2}$$

where $\mu_{i_{2D}}$ is the projection of μ_i onto image space and $\boldsymbol{\Sigma_{i_{2D}}}^{-1}$ is the inverse of the 2D covariance computed via the EWA Splatting approximation [55] of the perspectively projected 3D Gaussian. The model is optimized via stochastic gradient descent on image reconstruction loss:

$$L(\mathcal{G}|\phi, I_{gt}) = ||I_{\mathcal{G}}(\phi) - I_{gt}||_1 + L_{\text{D-SSIM}}(I_{\mathcal{G}}(\phi), I_{gt}), \tag{3}$$

where $I_{\mathcal{G}}(\phi)$ is the rendered image for pose ϕ. During training, the scene is periodically densified by cloning and splitting uncertain Gaussians and pruned by removing large and transparent Gaussians.

3.2 Deformable 3D Gaussian Splatting

A natural extension of 3DGS to dynamic scenes is to incorporate a learnable deformation field that models motion by deforming Gaussians over time. These deformable 3DGS methods typically learn a set of canonical 3D Gaussians $\mathcal{G}$ that represent the static scene geometry alongside a deformation network $\mathcal{D}$ that predicts time-dependent offset to transform each Gaussian at a given timestep. In addition to modeling the spatiotemporal trajectories of the means of the Gaussians, the deformation network $\mathcal{D}$ may also predict rotations and scales as functions time. The set of deformed Gaussians at time t, denoted by $\mathcal{G}_t$, can be formulated as:

$$(\mu + \Delta\mu, r + \Delta r, s + \Delta s) = \mathcal{D}(\mu, r, s, t), \tag{4}$$

where μ, r, and s are the canonical mean, rotation, and scaling parameters of the Gaussians, and $\Delta\mu$, Δr, and Δs are their predicted offsets time-dependent offsets. To render the scene at time t, the deformed Gaussians $\mathcal{G}_t$ are passed to the 3DGS differential rasterizer in the same way as a set of static Gaussians. During training, each camera pose ϕ is paired with its corresponding timestep t, parameterizing the rendering process as $I_{\mathcal{G}_t}(\phi)$.

Deformable 3D Gaussian Splatting (De3DGS) [50] implements the deformation network $\mathcal{D}$ as an MLP that predicts per-Gaussian trajectories. The amount of neural network inference scales with the number of Gaussians in the model, making it a natural setting to demonstrate the benefits of our proposed methods.

4 Method

Neural network inference is a major bottleneck in rendering speed for deformable 3DGS models. In this section, we propose two methods to reduce this cost. While our SpeeDe3DGS approach extends De3DGS, our methods are general and can be applied to any similar deformable 3DGS or 4DGS framework. Figure 2 presents a qualitative comparison of our methods on each dataset in 5.1.

GT	Baseline	Pruning w/o ASP	Pruning w/ ASP

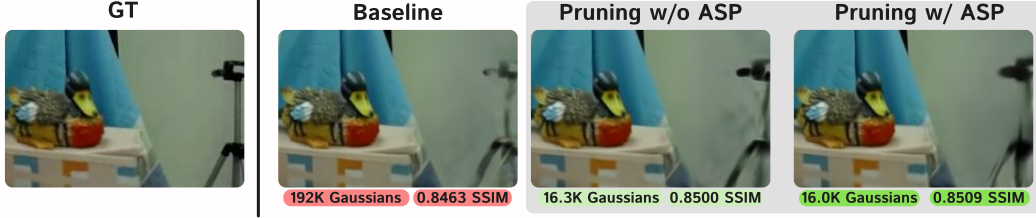192K Gaussians 0.8463 SSIM   16.3K Gaussians 0.8500 SSIM   16.0K Gaussians 0.8509 SSIM

Figure 3: **Comparison of our pruning methods on the real-world NeRF-DS *bell* scene.** Our pruned models achieve higher SSIM than the baseline De3DGS model while using $11\times$ fewer Gaussians. The left sides of the renderings appear visually identical. On the right sides of the renderings, pruning with ASP produces less temporal artifacts than both standard pruning and the unpruned baseline.

4.1 Temporal Pruning

Recent works have demonstrated that 3DGS models are heavily over-parameterized – comparable image quality can be achieved with significantly fewer primitives [17]. Several papers introduce pruning strategies to remove unimportant Gaussians from the model while preserving visual fidelity. Motivated by the observation that reducing the number of primitives directly lowers the inference load of the deformation network, we extend a recent pruning technique for static scenes to dynamic reconstruction. Additionally, we propose an annealing smooth pruning mechanism that improves the robustness of pruning scores to imprecise camera poses in dynamic real-world scenes.

4.1.1 Temporal Sensitivity Pruning Score

Our pruning approach builds directly on Speedy-Splat [16], which computes a per-Gaussian sensitivity score to evaluate the contribution of each Gaussian to scene reconstruction. At each pruning step, a fixed percentage of Gaussians with the lowest scores are removed. Specifically, the sensitivity pruning score $U_{\mathcal{G}_i}$ for Gaussian $\mathcal{G}_i$ is derived from its second-order sensitivity to the L_2 loss – computed as the Hessian of the L_2 loss – across all training views:

$$U_{\mathcal{G}_i} = \nabla^2_{g_i} L_2 = \sum_{\phi, t \in \mathcal{P}_{gt}} \left((\nabla_{g_i} I_{\mathcal{G}_t}(\phi))^2 + (I_{\mathcal{G}_t}(\phi) - I_{gt}) \nabla^2_{g_i} I_{\mathcal{G}_t}(\phi) \right), \qquad (5)$$

where g_i is the value of $\mathcal{G}_i$ projected onto image space given by Equation 2. When the L_1 loss approaches zero, the second term vanishes, leaving:

$$\tilde{U}_{\mathcal{G}_i} \approx \nabla^2_{g_i} L_2 \approx \sum_{\phi, t \in \mathcal{P}_{gt}} (\nabla_{g_i} I_{\mathcal{G}_t}(\phi))^2. \qquad (6)$$

Like in static scenes, we observe that the L_1 loss in dynamic scenes converges rapidly, making the second-order approximation effective in practice. Accordingly, we square and aggregate the gradient $\nabla_{g_i} I_{\mathcal{G}_t}(\phi)$, which is already computed in the renderer's backward pass, across all pixels and training views to obtain obtain per-Gaussian pruning scores $\tilde{U}_{\mathcal{G}_i}$.

Notably, because deformation updates vary acrosss timesteps, these gradients are inherently time-dependent and capture the second-order effects of each Gaussian *and its deformations* on the dynamic scene reconstruction. As such, our resulting **temporal sensitivity pruning score** extends Speedy-Splat's method to account for each Gaussian's contribution over time.

4.1.2 Annealing Smooth Pruning

Accurate pose estimation is a major challenge in many real-world datasets, especially in dynamic scenes where motion introduces additional ambiguity. Training under imprecise poses can cause models to overfit on specific frames, degrading generalization and leading to artifacts like spatial jitter between frames [35]. To improve robustness in these settings, De3DGS introduces an annealing smooth training (AST) mechanism that adds noise to the input timestamp in Equation 4 during training to regularize the deformation network:

$$(\mu + \Delta\mu, r + \Delta r, s + \Delta s) = \mathcal{D}(\mu, r, s, t + \mathcal{X}(i)), \qquad (7)$$
$$\mathcal{X}(i) = \mathbb{N}(0,1) \cdot \beta \cdot \Delta t \cdot (1 - i/\tau), \qquad (8)$$
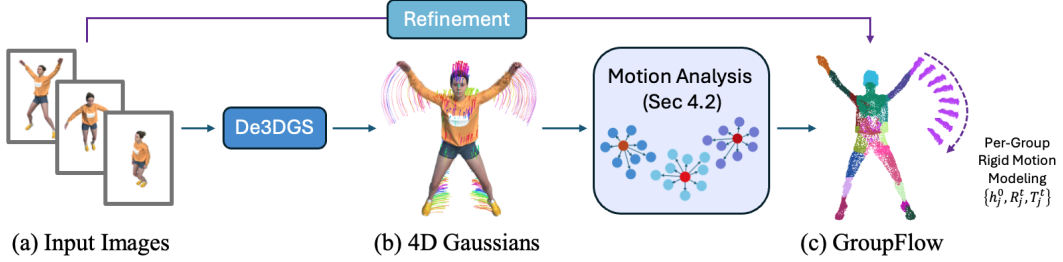
Figure 4: **Diagram of GroupFlow.** Given a 3D Gaussian model $\mathcal{G}$ and its deformations, we partition it into groups by identifying a subset of Gaussian as control points and then assigning each Gaussian to the control point h_j with the most similar motion trajectory. We then estimate the trajectory of each group via a rigid transformation $[R_j^t|T_j^t]$ at each time step. This drastically reduces inference by computing per-group instead of per-Gaussian deformations.

where $\mathcal{X}(i)$ is a linearly decaying Gaussian noise term applied at the i-th training iteration. Here, $\mathbb{N}(0,1)$ denotes standard normal noise, $\beta = 0.1$ is an empirically chosen scaling factor, Δt is the average time interval between frames, and $\tau = 20,000$ is the total number of iterations over which the noise is annealed. AST improves temporal generalization in early training, preserves fine details in dynamic objects by preventing over-smoothing in later stages, and reduces artifacts during interpolation tasks on real-world data.

We observe that pruning Gaussians from real-world scenes can also introduce temporal artifacts not present in the baseline model, indicating that sensitivity scores are susceptible to imprecise camera poses and may overfit to specific timesteps. To address this, we propose an **annealing smooth pruning (ASP)** mechanism, which adds the same linearly decaying Gaussian noise term $\mathcal{X}(i)$ to the input timestamp when rendering views for computing sensitivity scores. This small perturbation preserves the low L_1 residual error required by the Hessian-based approximation in Equation 6.

ASP mitigates artifacts introduced by pruning and can even reduce artifacts present in unpruned models. Temporal artifacts often appear only at specific timesteps and may go undetected when sensitivity scores are computed using fixed timestamps. By injecting noise into the timestamps, ASP lowers the temporal sensitivity scores of the Gaussians responsible for these artifacts, promoting their early removal while allowing for finer refinement later in training. Figure 3 illustrates an example where ASP produces cleaner results than both standard pruning and the unpruned baseline. ASP is ablated on the NeRF-DS dataset in Section 5.

4.2 GroupFlow

Many dynamic objects in real-world scenes exhibit approximately rigid motion. Motivated by this observation, we introduce **GroupFlow**, a grouping strategy that clusters Gaussians based on motion similarity and predicts a single shared trajectory for each group rather than separate deformations for each Gaussian. This reduces the number of trajectories that must be estimated by a neural network while capturing dominant motion patterns in the scene.

4.2.1 Flow Grouping via Motion Analysis

Our flow grouping method starts with dense deformable 3D Gaussian model $\mathcal{G}$ that captures the overall motion in the dynamic scene. Specifically, each deformable 3D Gaussian $\mathcal{G}_i$ is represented as a sequence of that Gaussian's means $\mu_i^t \in \mathbb{R}^3$ across F timesteps or frames:

$$\mathcal{M}_i = \{\mu_i^0, \mu_i^1, \ldots, \mu_i^{F-1}\}, \quad i \in \{1, \ldots, N\}, \tag{9}$$

where $N = |\mathcal{G}|$ is the total number of Gaussians. We denote the set of all Gaussian means across all timesteps as $\mathcal{M} = \{\mathcal{M}_i\}_1^N$. This unified 4D Gaussian representation enables similarity comparisons across trajectories.

We designate the first timestep $t = 0$ as the canonical frame to ensure temporal consistency. To form motion-based groups, we select J means $h_j^t \in \mathbb{R}^3$ as control points via farthest point sampling on the set of means at $t = 0$. All means μ_i are then assigned to the control point h_j with the most similar

trajectory obtained via:

$$\arg\min_{j\in\{1,\ldots,J\}} S_{i,j}, \tag{10}$$

$$S_{i,j} = \lambda_r \mathrm{std}_t(\|\mu_i^t - h_j^t\|) + (1-\lambda_r)\mathrm{mean}_t(\|\mu_i^t - h_j^t\|), \tag{11}$$

where $S_{i,j}$ is the trajectory similarity score between μ_i and control point h_j, $\mathrm{std}_t(\cdot)$ and $\mathrm{mean}_t(\cdot)$ denote the standard deviation and mean of time-varying residual $\mu_i^t - h_j^t \in \mathbb{R}^3$, and $\lambda_r = 0.5$ is an empirically selected weighting ratio. All means assigned to the control point h_j form a partitioning group $\mathcal{M}^j$ on $\mathcal{M}$.

Next, we estimate a time-varying rigid transformation for each group to capture its motion across time. For each group $j \in \{1, \ldots, J\}$ and time $t > 0$, we estimate an SE(3) rigid transformation $[R_j^t | T_j^t]$ that maps this group from the canonical frame at time $t = 0$ to time t using Umeyama alignment [39, 40]. Specifically, we randomly sample a subset of means $\mathcal{M}_{samp}^j$ ($|\mathcal{M}_{samp}^j| = \min\{N_j, N_{\max}\}$) from each group $\mathcal{M}^j$ and estimate their rigid transformation via:

$$\arg\min_{R_j^t, T_j^t} \sum_{\mu_i \in \mathcal{M}_{samp}^j} \|\mu_i^t - (R_j^t(\mu_i^0 - h_j^0) + h_j^0 + T_j^t)\|^2, \tag{12}$$

where μ_i^t is the mean at time t, μ_i^0 is the mean in the canonical frame at time $t = 0$, $N_j = |\mathcal{M}^j|$ is the number of Gaussians in group $\mathcal{M}^j$, and $N_{\max} = 100$ is an empirically selected threshold representing the maximum number of means sampled per group. These group-wise motion parameters $\{h_j^0, R_j^t, T_j^t\}$ represent the shared flow of the mean μ_i – and thus Gaussian $\mathcal{G}_i$ – within group $\mathcal{M}^j$ over time.

4.2.2 Flow Group Training and Inference

To predict the deformation of $\mu_i \in \mathcal{M}^j$ at timestep t, we apply an SE(3) rigid transformation $[R_j^t | T_j^t]$ to the canonical μ_i^0, relative to its assigned control point h_j:

$$\mu_i^t = R_j^t(\mu_i^0 - h_j^0) + h_j^0 + T_j^t. \tag{13}$$

We set the shared flow $\{h_j^0, R_j^t, T_j^t\}$ of each group $\mathcal{M}^j$ as learnable parameters. This proposed approach – **GroupFlow** – reduces the number of transformations predicted per timestep from N (one per Gaussian) to J (one per group) by enabling all Gaussians within a group to share the same motion parameters. The reduced inference load significantly accelerates both rendering speed and training time. We ablate rotation modeling and linear blend skinning in Appendix A.5

5 Experiments

5.1 Datasets

We evaluate our methods using seven real-world scenes from NeRF-DS [48], eight synthetic scenes from D-NeRF [37], and the four real-world scenes from the *vrig* subset of HyperNeRF [35]. For consistency across experiments, we use the COLMAP pose estimates and sparse point clouds provided by the dataset authors. Appendix A.2 reports metrics for the eight HyperNeRF scenes, many of which lack evaluation sets, that are evaluated in the Appendix of the De3DGS paper.

5.2 Implementation Details

Our SpeeDe3DGS pipeline integrates our pruning and GroupFlow methods into the De3DGS training pipeline. All scenes are trained for $30,000$ iterations. We adopt the pruning schedule from Speedy-Splat. Our pruning percentages – 80% during densification and 30% thereafter – are ablated in Appendix A.3. GroupFlow is initialized with $J = 200$ groups after densification completes at iteration $15,000$. The number of groups is ablated in Appendix A.4, and the inclusion of rotation modeling and linear blend skinning are ablated in Appendix A.5. Pruning score computation and grouping take seconds, and ASP adds negligible overhead [50]. Metrics are collected on an NVIDIA RTX A4000 GPU whenever possible – the sole exception is that the training times in Section 6.3 are collected on an NVIDIA RTX A6000. The reported results represent the average metrics across three independent runs for each scene.

7

Table 1: **Results on the real-world NeRF-DS Dataset.** The *P* column denotes the usage of pruning, *A* denotes ASP, *G* denotes GroupFlow, and *Size* denotes the combined size of the deformation network parameters and the .ply file. The best and second best metrics are highlighted. Our FPS and training times are collected on an A4000. Per-scene metrics are reported in Appendix A.1.

P	A	G	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	Size (MB) ↓	# Gaussians ↓	Train Time ↓
			23.79	0.8506	0.1786	35.38 (1.00×)	34.55 (1.00×)	133.60K (1.00×)	34.8m (1.00×)
✓			23.73	0.8475	0.1907	303.98 (8.47×)	4.68 (7.38×)	11.58K (11.25×)	16.2m (2.15×)
✓	✓		23.78	0.8509	0.1856	304.20 (8.60x)	4.62 (7.48×)	11.34K (11.78×)	16.0m (2.17x)
		✓	22.21	0.7862	0.2358	298.90 (8.45x)	34.87 (0.99x)	140.08K (0.95x)	21.4m (1.63x)
✓	✓	✓	23.42	0.8449	0.1944	366.73 (10.37x)	4.48 (7.71×)	11.51K (11.61x)	12.9m (2.71x)

Table 2: **Results on the synthetic D-NeRF dataset.** Pruning is performed without ASP. *Size* denotes the combined size of the deformation parameters and the .ply file. The best and second best metrics are highlighted. Our FPS and training times are collected on an A4000. Per-scene metrics and numbers of Gaussians for our methods are reported in Appendix A.1.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	Size (MB) ↓	Train Time ↓
D-NeRF [37]	29.67	0.95	0.08	0.1	-	-
TiNeuVox-B [9]	32.67	0.97	0.04	1.5	48	28m
K-Planes [10]	32.61	0.97	-	0.97	418	52m
HexPlane [4]	31.04	0.97	0.04	2.5	38	11.5m
FFDNeRF [15]	32.68	0.97	0.04	<1	-	-
4DGS (CVPR) [45]	34.05	0.98	0.02	82	18	20m
4DGS (ICLR) [49]	32.99	0.97	0.03	376	278	-
C-D3DGS [20]	32.19	0.97	0.04	150	159	8m
SC-GS [18]	43.31	0.99	0.01	295	-	-
De3DGS [50] (Baseline)	38.90	0.9891	0.0143	97.38 (1.00x)	16.83 (1.00x)	22.1m (1.00x)
+ Pruning (Ours)	36.23	0.9795	0.0344	383.91 (3.94x)	2.89 (5.82×)	11.5m (1.92x)
+ GroupFlow (Ours)	36.35	0.9852	0.0181	393.47 (4.04x)	16.84 (1.00×)	15.2m (1.46x)
+ Both (Ours)	34.83	0.9767	0.0371	409.29 (4.20x)	2.89 (5.82×)	10.6m (2.07x)

6 Results

6.1 Evaluation on NeRF-DS

As shown in Table 1, SpeeDe3DGS accelerates the inference and training speed of the NeRF-DS dataset by $10.37\times$ and $2.71\times$, respectively. Pruning with ASP consistently achieves better image quality, fewer primitives, and faster inference and training speed than the standard pruning method. Models pruned with ASP maintain comparable image quality to the unpruned baseline despite using $11.78\times$ fewer Gaussians. Figure 3 illustrates an example where ASP yields cleaner results than both the standard pruning method and the unpruned baseline. Interestingly, GroupFlow performs better when used in conjunction with pruning – Gaussian motion in real scenes is often noisy, so pruning may reduce the amount of noise in the motion analysis and transformation optimization.

6.2 Evaluation on D-NeRF

On the synthetic D-NeRF dataset, presented in Table 2, SpeeDe3DGS improves rendering speed by $4.20\times$, compresses model size by $12.95\times$, and shortens training time by $2.07\times$. ASP is not used in these experiments because synthetic scenes provide exact ground-truth camera poses. GroupFlow outperforms pruning in both image quality and rendering speed, likely due to the low-noise nature of Gaussian motion in synthetic environments. SpeeDe3DGS produces smaller model sizes, faster rendering speeds, and shorter training times than all compared methods.

6.3 Evaluation on HyperNeRF

Table 3 reports results for the four scenes in the *vrig* subset of the real-world HyperNeRF dataset. SpeeDe3DGS accelerates accelerates rendering speed by a drastic $58.23\times$ while compressing model size by $18.82\times$ and contracting training time by $5.22\times$ – far surpassing all compared methods. When applied individually, pruning and GroupFlow produce much smaller rendering speed-ups of $16.06\times$ and $21.93\times$, respectively, emphasizing the complementary benefits of combining our proposed

Table 3: **Method comparison on the real-world HyperNeRF "vrig" dataset.** Pruning is performed with ASP. *Size* denotes the combined size of the deformation parameters and the .ply file. LPIPS is not reported by the other methods. The best and second best metrics are highlighted. Our FPSs are collected on an A4000 and our training times are collected on an A6000 due to memory constraints. Per-scene metrics and numbers of Gaussians for our methods are reported in Appendix A.1, and Appendix A.2 records metrics for eight additional HyperNeRF scenes.

Method	PSNR ↑	SSIM ↑	FPS ↑ (A4000)	Size (MB) ↓	Train Time ↓ (A6000)
Nerfies [34]	22.2	0.803	<1	-	∼h
HyperNeRF [35]	22.4	0.814	<1	-	32h
TiNeuVox-B [9]	24.3	0.836	1	48	30m
3D-GS [21]	19.7	0.680	55	52	40m
FFDNeRF [15]	24.2	0.842	0.05	440	-
4DGS (CVPR) [45]	25.2	0.845	34	61	1̃h
E-D3DGS [3]	25.43	0.697	139.3	33	1h15m
De3DGS [50] (Baseline)	22.68	0.6118	7.42 (1.00x)	222.79 (1.00×)	1h51.3m (1.00×)
+ Pruning (Ours)	22.50	0.6001	119.22 (16.06×)	13.15 (16.94×)	27.9m (3.99×)
+ GroupFlow (Ours)	21.00	0.5761	162.81 (21.93×)	205.23 (1.09×)	58.8m (1.89×)
+ Both (Ours)	21.08	0.5821	432.33 (58.23×)	11.84 (18.82×)	21.3m (5.22×)

techniques. Metrics for the eight HyperNeRF scenes evaluated in the Appendix of De3DGS are reported in Appendix A.2.

Training the baseline *banana* scene exceeds the memory capacity of the NVIDIA RTX A4000 GPU used for our other experiments, so we train these scenes on an NVIDIA RTX A6000 GPU and subsequently collect rendering speeds on the A4000. Notably, pruning enables all scenes to train succesfully on the A4000. The HyperNeRF dataset is known to have imprecise camera poses that negatively affect visual quality [27, 35, 50].

7   Limitations

As discussed in Section 4, our methods are general and can be applied any similar deformable 3DGS or 4DGS framework. We choose to extend De3DGS to highlight the benefits of our method, but this choice also inherits the limitations of De3DGS. For instance, De3DGS struggles to fit scenes with noisy camera poses like the ones in the HyperNeRF dataset [35, 50], so our pipeline exhibits similar behavior in Section 6.3. The motion analysis in GroupFlow similarly relies on the quality of the Gaussian trajectories modeled by the dynamic representation, but De3DGS does not incorporate strong motion priors like the methods in Section 2.2. Additionally, since the baseline D-NeRF scenes already contain few Gaussians, aggressive pruning ratios can sometimes degrade reconstruction results. While integrating our methods into a different framework could potentially yield even better results, Section 5 demonstrates that our SpeeDe3DGS pipeline already outperforms existing methods in rendering speed, training time, and model size, while producing comparable image quality.

8   Conclusion

In this paper, we present Speedy Deformable 3D Gaussian Splatting (SpeeDe3DGS), a method that accelerates dynamic 3DGS representations by reducing neural inference through two complementary techniques. First, we propose a temporal sensitivity pruning score that identifies and removes Gaussians with low contribution to dynamic scene reconstruction, eliminating their inference overhead. We improves the robustness of pruning to imprecise camera poses in real-world scenes with a novel annealing smooth pruning mechanism. Second, we introduce GroupFlow, a motion analysis method that clusters Gaussians with similar trajectories and predicts a single shared motion per group instead of per Gaussian. Combined, these techniques accelerate rendering by $10.38\times$, compress model size by $7.71\times$, and reduce training time by $2.71\times$ on the real-world NeRF-DS dataset. Our SpeeDe3DGS methods are general and can be applied to any deformable 3DGS or 4DGS framework.

9   Acknowledgments

References

[1] Muhammad Salman Ali, Sung-Ho Bae, and Enzo Tartaglione. Elmgs: Enhancing memory and computation scalability through compression for 3d gaussian splatting. *arXiv preprint arXiv:2410.23213*, 2024.

[2] Muhammad Salman Ali, Maryam Qamar, Sung-Ho Bae, and Enzo Tartaglione. Trimming the fat: Efficient compression of 3d gaussian splats through pruning. *arXiv preprint arXiv:2406.18214*, 2024.

[3] Jeongmin Bae, Seoha Kim, Youngsik Yun, Hahyun Lee, Gun Bang, and Youngjung Uh. Per-gaussian embedding-based deformation for deformable 3d gaussian splatting. In *European Conference on Computer Vision*, pages 321–335. Springer, 2024.

[4] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. pages 130–141, 2023.

[5] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. pages 14324–14334, 2021.

[6] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Light-gaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023.

[7] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al. Light-gaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37:140138–140158, 2024.

[8] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*, pages 165–181. Springer, 2024.

[9] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. 2022.

[10] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. pages 12479–12488, 2023.

[11] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. 2021.

[12] Quankai Gao, Qiangeng Xu, Hao Su, Ulrich Neumann, and Zexiang Xu. Strivec: Sparse tri-vector radiance fields. pages 17569–17579, 2023.

[13] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. *arXiv preprint arXiv:2312.04564*, 2023.

[14] Xiang Guo, Guanying Chen, Yuchao Dai, Xiaoqing Ye, Jiadai Sun, Xiao Tan, and Errui Ding. Neural deformable voxel grid for fast optimization of dynamic view synthesis. 2022.

[15] Xiang Guo, Jiadai Sun, Yuchao Dai, Guanying Chen, Xiaoqing Ye, Xiao Tan, Errui Ding, Yumeng Zhang, and Jingdong Wang. Forward flow for novel view synthesis of dynamic scenes. pages 16022–16033, 2023.

[16] Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-splat: Fast 3d gaussian splatting with sparse pixels and sparse primitives. *arXiv*, 2024.

[17] Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. *arXiv*, 2024.

[18] 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. *arXiv preprint arXiv:2312.14937*, 2023.

[19] Saqib Javed, Ahmad Jarrar Khan, Corentin Dumery, Chen Zhao, and Mathieu Salzmann. Temporally compressed 3d gaussian splatting for dynamic scenes, 2024.

[20] Kai Katsumata, Duc Minh Vo, and Hideki Nakayama. A compact dynamic 3d gaussian representation for real-time dynamic view synthesis. In *European Conference on Computer Vision*, pages 394–412. Springer, 2024.

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

[22] Agelos Kratimenos, Jiahui Lei, and Kostas Daniilidis. Dynmf: Neural motion factorization for real-time dynamic view synthesis with 3d gaussian splatting. In *European Conference on Computer Vision*, pages 252–269. Springer, 2024.

[23] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21719–21728, 2024.

[24] Weikai Lin, Yu Feng, and Yuhao Zhu. Rtgs: Enabling real-time gaussian splatting on mobile devices using efficiency-guided pruning and foveated rendering. *arXiv preprint arXiv:2407.00435*, 2024.

[25] Hengyu Liu, Yifan Liu, Chenxin Li, Wuyang Li, and Yixuan Yuan. Lgs: A light-weight 4d gaussian splatting for efficient surgical scene reconstruction, 2024.

[26] Jia-Wei Liu, Yan-Pei Cao, Weijia Mao, Wenqiao Zhang, David Junhao Zhang, Jussi Keppo, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. Devrf: Fast deformable voxel radiance fields for dynamic scenes. *arXiv preprint arXiv:2205.15723*, 2022.

[27] Qingming LIU, Yuan Liu, Jiepeng Wang, Xianqiang Lyu, Peng Wang, Wenping Wang, and Junhui Hou. MoDGS: Dynamic gaussian splatting from casually-captured monocular videos with depth priors. In *The Thirteenth International Conference on Learning Representations*, 2025.

[28] Xiangrui Liu, Xinju Wu, Pingping Zhang, Shiqi Wang, Zhu Li, and Sam Kwong. Compgs: Efficient 3d scene representation via compressed gaussian splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 2936–2944, 2024.

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

[30] Marko Mihajlovic, Sergey Prokudin, Marc Pollefeys, and Siyu Tang. Resfields: Residual neural fields for spatiotemporal signals. *International Conference on Learning Representations*, 2024.

[31] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[32] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv preprint arXiv:2403.13806*, 2024.

[33] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–17, 2024.

[34] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.

[35] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021.

[36] Sida Peng, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Representing volumetric videos as dynamic mlp maps. 2023.

[37] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. 2020.

[38] Sara Sabour, Lily Goli, George Kopanas, Mark Matthews, Dmitry Lagun, Leonidas Guibas, Alec Jacobson, David J. Fleet, and Andrea Tagliasacchi. SpotLessSplats: Ignoring distractors in 3d gaussian splatting. *arXiv:2406.20055*, 2024.

[39] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, pages 109–116. Citeseer, 2007.

[40] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04):376–380, 2024.

[41] Diwen Wan, Ruijie Lu, and Gang Zeng. Superpoint gaussian splatting for real-time high-fidelity dynamic scene reconstruction. In *Proceedings of the 41st International Conference on Machine Learning*, pages 49957–49972, 2024.

[42] Chaoyang Wang, Ben Eckart, Simon Lucey, and Orazio Gallo. Neural trajectory fields for dynamic novel view synthesis. 2021.

[43] Feng Wang, Zilong Chen, Guokang Wang, Yafei Song, and Huaping Liu. Masked space-time hash encoding for efficient dynamic scene reconstruction. *Advances in neural information processing systems*, 2023.

[44] Rui Wang, Quentin Lohmeyer, Mirko Meboldt, and Siyu Tang. Degauss: Dynamic-static decomposition with gaussian splatting for distractor-free 3d reconstruction, 2025.

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

[46] Jiahao Wu, Rui Peng, Zhiyan Wang, Lu Xiao, Luyang Tang, Jinbo Yan, Kaiqiang Xiong, and Ronggang Wang. Swift4d: Adaptive divide-and-conquer gaussian splatting for compact and efficient reconstruction of dynamic scene. *arXiv preprint arXiv:2503.12307*, 2025.

[47] Kai Xu, Tze Ho Elden Tse, Jizong Peng, and Angela Yao. Das3r: Dynamics-aware gaussian splatting for static scene reconstruction. *arXiv preprint arxiv:2412.19584*, 2024.

[48] Zhiwen Yan, Chen Li, and Gim Hee Lee. Nerf-ds: Neural radiance fields for dynamic specular objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8285–8295, 2023.

[49] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. 2024.

[50] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023.

[51] Yuxin Yao, Zhi Deng, and Junhui Hou. Riggs: Rigging of 3d gaussians for modeling articulated objects in videos. *arXiv preprint arXiv:2503.16822*, 2025.

[52] Xinjie Zhang, Zhening Liu, Yifan Zhang, Xingtong Ge, Dailan He, Tongda Xu, Yan Wang, Zehong Lin, Shuicheng Yan, and Jun Zhang. Mega: Memory-efficient 4d gaussian splatting for dynamic scenes, 2024.

[53] Zhaoliang Zhang, Tianchen Song, Yongjae Lee, Li Yang, Cheng Peng, Rama Chellappa, and Deliang Fan. Lp-3dgs: Learning to prune 3d gaussian splatting, 2024.

[54] Hongyu Zhou, Jiahao Shao, Lu Xu, Dongfeng Bai, Weichao Qiu, Bingbing Liu, Yue Wang, Andreas Geiger, and Yiyi Liao. Hugs: Holistic urban 3d scene understanding via gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21336–21345, June 2024.

[55] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.

A Appendix

A.1 Per-Scene Metrics

PSNR, SSIM, LPIPS, FPS, and training times for each scene from the NeRF-DS, D-NeRF, and HyperNeRF *vrig* datasets are recorded in Tables 4, 5, 6, 7, and Table 8, respectively. The experimental setup is outlined in Section 5.2; the reported metrics are averaged over three independent runs to reduce variance. Pruning is performed with ASP on the NeRF-DS and HyperNeRF datasets. All metrics are collected with an NVIDIA RTX A4000 GPU with the exception of training time on the HyperNeRF dataset, which is collected with an A6000 due to memory constraints.

Table 4: **PSNR ↑ on each scene.** *P* denotes the usage of pruning and *G* denotes GroupFlow.

P	G	as	basin	bell	cup	plate	press	sieve	balls	warrior	hook	jacks	lego	mutant	standup	trex	broom	printer	chicken	banana
		25.94	19.67	25.20	24.48	20.32	25.53	25.39	41.04	41.17	36.60	37.29	32.00	41.93	43.43	37.72	20.41	20.49	22.90	26.90
✓		25.96	19.63	25.48	24.59	20.38	25.38	25.08	37.96	39.44	34.37	35.50	30.18	37.39	39.18	35.82	20.43	20.26	22.78	26.52
	✓	25.30	19.30	19.33	23.82	17.83	24.71	25.18	37.61	40.13	34.67	34.87	29.97	39.38	39.64	34.60	18.94	20.96	22.90	21.19
✓	✓	25.64	19.42	25.01	23.88	19.86	24.75	25.20	36.69	38.79	32.97	33.52	29.52	36.36	37.15	33.61	18.76	20.96	23.11	21.50

Table 5: **SSIM ↑ on each scene.** *P* denotes the usage of pruning and *G* denotes GroupFlow.

P	G	as	basin	bell	cup	plate	press	sieve	balls	warrior	hook	jacks	lego	mutant	standup	trex	broom	printer	chicken	banana
		0.8823	0.7929	0.8463	0.8866	0.8108	0.8625	0.8729	0.9956	0.9866	0.9848	0.9892	0.9753	0.9942	0.9939	0.9930	0.3213	0.6487	0.6100	0.8673
✓		0.8801	0.7936	0.8509	0.8890	0.8104	0.8613	0.8710	0.9922	0.9788	0.9731	0.9818	0.9584	0.9799	0.9852	0.9868	0.2909	0.6423	0.6120	0.8553
	✓	0.8630	0.7762	0.5585	0.8743	0.7163	0.8514	0.8636	0.9936	0.9839	0.9803	0.9855	0.9695	0.9915	0.9903	0.9894	0.2359	0.6570	0.6203	0.7913
✓	✓	0.8696	0.7871	0.8420	0.8812	0.8054	0.8551	0.8668	0.9911	0.9771	0.9683	0.9781	0.9555	0.9775	0.9823	0.9836	0.2413	0.6624	0.6279	0.7970

Table 6: **LPIPS ↓ on each scene.** *P* denotes the usage of pruning and *G* denotes GroupFlow.

P	G	as	basin	bell	cup	plate	press	sieve	balls	warrior	hook	jacks	lego	mutant	standup	trex	broom	printer	chicken	banana
		0.1832	0.1869	0.1596	0.1587	0.2210	0.1914	0.1495	0.0086	0.0265	0.0176	0.0143	0.0215	0.0069	0.0089	0.0104	0.5171	0.2500	0.2077	0.1381
✓		0.1924	0.1966	0.1635	0.1595	0.2312	0.1985	0.1571	0.0188	0.0534	0.0422	0.0291	0.0489	0.0333	0.0262	0.0236	0.6672	0.2724	0.2301	0.1603
	✓	0.2096	0.2150	0.3389	0.1788	0.3341	0.2104	0.1637	0.0112	0.0308	0.0220	0.0181	0.0271	0.0099	0.0121	0.0130	0.7114	0.2610	0.2287	0.2140
✓	✓	0.2073	0.2074	0.1718	0.1723	0.2383	0.2085	0.1640	0.0204	0.0567	0.0466	0.0325	0.0508	0.0358	0.0288	0.0253	0.7168	0.2742	0.2304	0.2146

Table 7: **FPS ↑ on each scene.** Speed-ups ↑ are reported in (parentheses). *P* denotes the usage of pruning and *G* denotes GroupFlow. Metrics are collected on an NVIDIA RTX A4000 GPU.

P	G	as	basin	bell	cup	plate	press	sieve	balls	warrior	hook	jacks	lego	mutant	standup	trex	broom	printer	chicken	banana
		39.89	33.81	24.00	36.53	36.87	38.51	38.05	63.61	210.10	75.99	131.88	40.40	60.98	148.34	47.77	3.32	14.28	8.87	3.22
		(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)
✓		341.59	305.66	234.88	298.36	320.10	322.56	306.22	426.35	374.97	368.02	362.88	330.31	369.87	485.08	353.83	166.67	147.19	111.83	51.19
✓		(8.56×)	(9.04×)	(9.79×)	(8.17×)	(8.68×)	(8.38×)	(8.05×)	(6.70×)	(1.78×)	(4.84×)	(2.75×)	(8.18×)	(6.07×)	(3.27×)	(7.41×)	(50.15×)	(10.31×)	(12.61×)	(15.88×)
	✓	297.10	277.93	316.34	296.29	319.97	304.39	280.29	283.70	597.41	368.03	462.07	282.49	389.60	458.82	305.67	163.99	215.30	175.84	96.09
	✓	(7.45×)	(8.22×)	(13.18×)	(8.11×)	(8.68×)	(7.90×)	(7.37×)	(4.46×)	(2.84×)	(4.84×)	(3.50×)	(6.99×)	(6.39×)	(3.09×)	(6.40×)	(49.34×)	(15.08×)	(19.82×)	(29.80×)
✓	✓	357.41	332.12	358.51	359.96	343.21	359.87	365.83	329.20	398.49	484.96	488.86	437.27	375.01	389.12	371.42	494.61	432.94	443.29	358.47
✓	✓	(8.96×)	(9.82×)	(14.94×)	(9.85×)	(9.31×)	(9.35×)	(9.61×)	(5.18×)	(1.90×)	(6.38×)	(3.71×)	(10.82×)	(6.15×)	(2.62×)	(7.78×)	(148.82×)	(30.32×)	(49.97×)	(111.18×)

Table 8: **Training time ↓ in minutes on each scene.** Speed-ups ↑ are reported in (parentheses). *P* denotes the usage of pruning and *G* denotes GroupFlow. Metrics for the NeRF-DS and D-NeRF datasets are collected on an NVIDIA RTX A4000 GPU; metrics for the HyperNeRF dataset are collected on an RTX A6000.

P	G	NeRF-DS (A4000) as	basin	bell	cup	plate	press	sieve	D-NeRF (A4000) balls	warrior	hook	jacks	lego	mutant	standup	trex	HyperNeRF vrig (A6000) broom	printer	chicken	banana
		31.66	35.44	44.90	34.54	31.90	32.49	32.81	26.53	10.97	20.74	13.90	36.46	24.30	13.47	30.18	164.44	42.64	64.75	173.21
		(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)	(1.00×)
✓		15.15	16.57	20.50	16.17	13.57	14.89	15.42	11.93	8.22	11.28	8.91	16.63	12.56	8.51	13.93	26.25	15.28	20.81	49.24
✓		(2.09×)	(2.14×)	(2.19×)	(2.14×)	(2.35×)	(2.18×)	(2.13×)	(2.22×)	(1.33×)	(1.84×)	(1.56×)	(2.19×)	(1.94×)	(1.58×)	(2.17×)	(6.27×)	(2.79×)	(3.11×)	(3.52×)
	✓	18.85	21.63	25.63	20.49	21.28	21.18	20.65	18.33	10.43	13.86	10.85	22.69	16.08	10.64	18.56	64.34	24.72	39.12	107.20
	✓	(1.68×)	(1.64×)	(1.75×)	(1.69×)	(1.50×)	(1.53×)	(1.59×)	(1.45×)	(1.05×)	(1.50×)	(1.28×)	(1.61×)	(1.51×)	(1.27×)	(1.63×)	(2.56×)	(1.72×)	(1.66×)	(1.62×)
✓	✓	13.07	13.65	15.83	14.04	11.17	12.61	12.43	11.51	8.38	10.11	8.42	14.41	11.34	8.66	12.32	21.25	11.46	14.20	38.42
✓	✓	(2.42×)	(2.60×)	(2.84×)	(2.46×)	(2.86×)	(2.58×)	(2.64×)	(2.31×)	(1.31×)	(2.05×)	(1.65×)	(2.53×)	(2.14×)	(1.55×)	(2.45×)	(7.74×)	(3.72×)	(4.56×)	(4.51×)

Table 9: **Results on the eight real-world HyperNeRF scenes that are evaluated in the De3DGS paper [50].** *P* denotes the usage of pruning, *G* denotes GroupFlow, and *Size* denotes the combined size of the deformation parameters and the .ply file. LPIPS is not reported by the other methods. The best and second best metrics are highlighted. Our FPS measurements are collected on an A4000 and our training times are collected on an A6000 due to memory constraints.

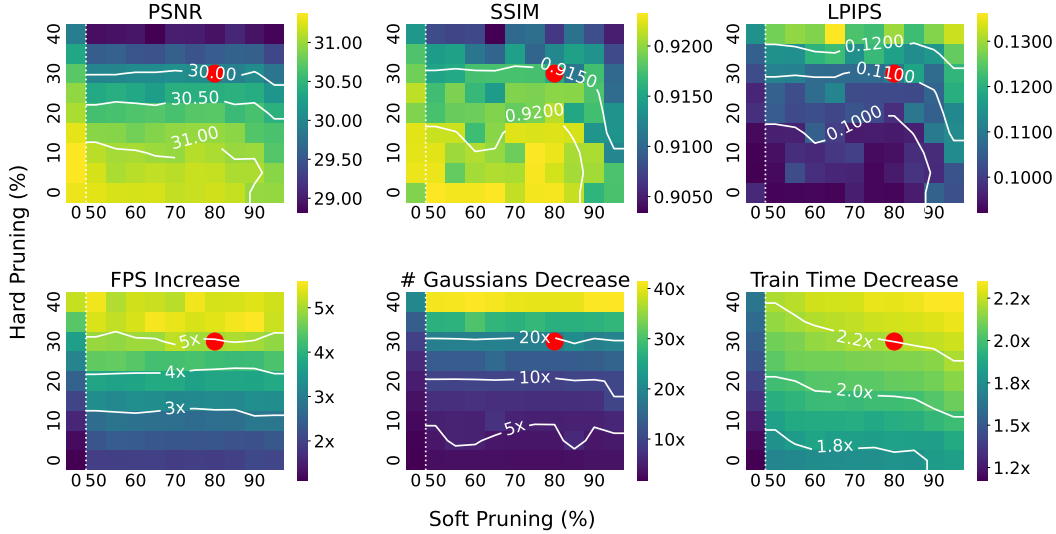| P	G	FPS ↑ (A4000)	Size (MB) ↓	# Gaussians ↓	Train Time ↓ (A6000)
		9.55 (1.00×)	151.10 (1.00×)	630.67K (1.00×)	82.0 (1.00×)
✓		104.36 (10.92×)	13.68 (11.05×)	49.63K (12.71×)	28.5 (2.88×)
	✓	148.58 (15.55×)	150.91 (1.00×)	630.72K (1.00×)	56.0 (1.47×)
✓	✓	319.60 (33.46×)	13.34 (11.32×)	49.06K (12.86×)	22.7 (3.62×)

Figure 5: We sweep pruning percentages in 5% increments for densification-stage pruning (0%, 50% − 95%) and post-densification pruning (0% − 40%) on all scenes listed in Section 5.1. Experiments are performed 3× on each scene without ASP or GroupFlow; the reported metrics are averaged across all runs. (0%, 0%) is the baseline 3DGS model, the first row (:, 0%) is densification-stage pruning in isolation, and the first column (0%, :) is post-densification pruning in isolation. The red dots at (80%, 30%) denote our selected percentage settings. We report the FPS increase and the Number of Gaussians and Train Time decrease factors to be consistent with the format in Table 1.

A.2 Additional HyperNeRF Evaluation

Table 9 records FPS, size in MB, number of Gaussians, and training time in minutes for eight real-world HyperNeRF scenes: *espresso*, *americano*, *cookie*, *chicken*, *torchocolate*, *lemon*, *hand*, and *printer*. Following De3DGS [50], image quality metrics are not reported because most scenes lack evaluation sets. FPS is collected on an NVIDIA RTXA4000 and training time is collected on an A6000 due to memory constraints. Our methods substantially improve inference speed, model size, and training time.

A.3 Ablation on Pruning Percentages

We follow the pruning schedule proposed in Speedy-Splat [16]. In Figure 5, we perform a parameter sweep over densification-stage and post-densification pruning percentages in 5% intervals. We conduct each experiment 3× on the NeRF-DS and D-NeRF scenes listed in Section 5.1 to reduce variance, then average the metrics across all runs. All experiments are run without ASP and GroupFlow to isolate the effects of our standard pruning method in isolation. Our (80%, 30%) pruning percentages are empirically selected to produce a favorable balance between speed and quality.

15

Table 10: **Group count J on the D-NeRF dataset.** The *Params* column reports deformation parameter size in MB, and *Train* reports training time in minutes. The parameters for our selected $J = 200$ groups uses less memory than the De3DGS MLP. Accurate measurements are collected by averaging metrics across three independent runs to reduce variance.

J	PSNR ↑	SSIM ↑	LPIPS ↓	Params ↓	FPS ↑	Train ↓
Baseline	38.90	0.9891	0.0143	**1.94 (1.00×)**	97.38 (1.00×)	22.1 (1.00×)
50	35.23	0.9829	0.0208	0.80 (2.43×)	393.84 (4.04×)	15.33 (1.44×)
100	36.00	0.9846	0.0189	1.11 (1.74×)	392.87 (4.03×)	15.15 (1.46×)
200	36.35	0.9852	0.0181	**1.74 (1.12×)**	393.47 (4.04×)	15.18 (1.46×)
500	36.78	0.9861	0.0173	3.62 (0.54×)	396.45 (4.07×)	15.08 (1.47×)
1000	36.96	0.9864	0.0171	6.75 (0.29×)	397.16 (4.08×)	15.09 (1.46×)

Table 11: **Ablation on GroupFlow design choices in the synthetic D-NeRF dataset.** The rotation offset modeling and linear blend skinning (LBS) variants are ablated independently.

Method	PSNR ↑	SSIM ↑	LPIPS ↓	Params ↓	FPS ↑	Train ↓
Baseline	38.90	0.9891	0.0143	1.94 (1.00×)	97.38 (1.00×)	22.1 (1.00×)
+ GroupFlow	36.35	0.9852	0.0181	1.74 (1.12×)	393.47 (4.04×)	15.18 (1.46×)
w/ Rotation	36.22	0.9849	0.0178	1.74 (1.12×)	236.87 (2.43×)	15.54 (1.42×)
w/ LBS	33.89	0.9799	0.0243	1.74 (1.12×)	328.76 (3.38×)	19.04 (1.16×)

A.4 Ablation on Number of Groups

Table 10 ablates the number of groups J on the D-NeRF dataset. We find that increasing J yields higher image quality by allowing more degrees of freedom for the grouped flows to fit the scene's motion. Despite varying group counts, rendering speed and training time are similar across the experiments due to efficient parallelization. However, the number of learnable parameters scales linearly with J. We select $J = 200$ for our experiments to maintain image quality while using 1.74MB of parameters – less than the 1.94MB required by the baseline De3DGS deformation MLP. Notably, even with $J = 1000$, the number of parameters remains lightweight at just 6.75MB, yielding higher image quality than our chosen configuration with minimal additional overhead.

A.5 Ablation on GroupFlow Design Choices

Table 11 ablates alternative design choices for GroupFlow on the D-NeRF dataset. Specifically, we consider two variants: **rotation offset modeling**, where each Gaussian rotates relative to its group, and **linear blend skinning (LBS)**, where each Gaussian is influenced by multiple neighboring control points. Detailed descriptions of these methods are provided in the subsections below. All experiments use $J = 200$ groups, with results averaged over three independent runs. Neither variant improves image quality, and both incur additional computational overhead that slows rendering and training. These findings underscore the effectiveness of our design choices in our GroupFlow method described in Section 4.2, which achieves the best performance across all metrics.

A.5.1 Rotation Offset Modeling

Our GroupFlow method assigns each Gaussian a fixed rotation quaternion $r \in \mathbb{R}^4$. An alternative design is to model the Gaussian's rotation as a time-varying offset:

$$r_i^t = R_j^t r_i^0, \tag{14}$$

where R_j^t is the rotation matrix of group j at time t, and r_i^0 is the canonical rotation of Gaussian $\mathcal{G}_i$. We parameterize R_j^t using a Lie algebra ($\mathfrak{so}(3)$) 3D vector representation to avoid the redundancy of explicit matrices or quaternions. However, applying Equation 14 requires converting both R_j^t and r_i^0 to matrix form before multiplication, resulting in additional computational overhead and, as shown in Table 11, slower inference.

A.5.2 Linear Blend Skinning

SC-GS [18] adopts Linear Blend Skinning (LBS) to model non-rigid motion, allowing each Gaussian $\mathcal{G}_i$ to be influenced by a set of neighboring control points $\mathcal{N}_i$ rather than a single control point:

$$\mu_i^t = \sum_{k \in \mathcal{N}_i} w_{ik} R_k^t (\mu_i^0 - h_k^0) + h_k^0 + T_k^t, \tag{15}$$

where w_{ik} denotes the influence weight of control point k on Gaussian $\mathcal{G}_i$.

$$w_{ik} = \frac{\hat{w}_{ik}}{\sum_{k \in \mathcal{N}_i} \hat{w}_{ik}}, \text{ where } \hat{w}_{ik} = \exp(-\frac{d_{ik}^2}{2\sigma_k^2}), \tag{16}$$

where d_{ik} is the distance between mean of Gaussian μ_i and the neighboring control point h_k^0, σ_k is the learned radius parameter of h_k^0. While LBS enables more flexible deformations, it introduces sensitivity to the choice of neighborhood size $|\mathcal{N}_i| = 5$ and can struggle with handling complex topological variations. Furthermore, as reported in Table 11, it incurs additional computational overhead that slows down inference.