TreeSplat: Mergeable Tree for Deformable Gaussian Splatting

Qiuhong Shen[1] **Xingyi Yang**[2,1] **Xinchao Wang**[1*]
[1]National University of Singapore [2]The Hong Kong Polytechnic University
qiuhong.shen@u.nus.edu, xingyi.yang@polyu.edu.hk, xinchao@nus.edu.sg

Abstract

Dynamic 3D scene reconstruction from multi-view videos demands representation to model complex deformations at scale. Current Gaussian Splatting based methods often either suffer from significant computation cost due to dense MLP-based modeling or explicit modeling deformation of each Gaussian independently. However, the dynamics of objects within a scene are typically hierarchical and exhibit structural correlations. To leverage these structural priors into the representation, we introduce **TreeSplat**, a **Tree** data structure for deformable Gaussian **Splat**ting. In TreeSplat, as the name suggests, motions of Gaussian are represented hierarchically within a tree. Each node learns coefficients for time-varying basis functions, defining a part of the motion. The full motion for any given Gaussian is then determined by accumulating these transformations along the tree path from its leaf node to the root node. This tree isn't predefined; instead, it is constructed adaptively alongside Gaussian densification, where cloning or splitting a Gaussian correspondingly creates new leaf nodes. One central property of TreeSplat is its mergeability; after optimization during training, the hierarchical motion parameters for each Gaussian can be efficiently consolidated. By performing this merging step before test time, we eliminate the need to traverse the tree explicitly for each Gaussian during rendering. This results in dramatically faster rendering over 200 FPS and compact storage, while maintaining state-of-the-art rendering quality. Experiments on diverse synthetic and real-world datasets validate these advantages. Code and results are available at `https://github.com/florinshen/treesplat`.

1 Introduction

Dynamic 3D scene reconstruction aims to recover deformable 3D representations from multi-view videos. This, in turn, makes it possible to synthesize novel views from different viewpoints and moments in time. Despite recent progress, achieving efficient dynamic 3D scene reconstruction remains challenging due to limitations in fitting time, data storage, and rendering speed.

Neural Radiance Fields (NeRF) [1–6] have demonstrated remarkable performance in static 3D reconstruction by optimizing multi layer perceptrons (MLPs) with volumetric ray marching. Several extensions [7–9, 9–18] incorporated temporal components into the MLP to capture scene deformations over time. However, these methods often suffer from high computational demands, significantly impacting their efficiency, particularly when scaling to complex dynamic scenes.

Recent developments in 3D Gaussian Splatting (3DGS) [19] have established it as a compelling alternative for efficient 3D scene reconstruction. Building upon this foundation, a growing body of work [16, 20–37] has incorporated temporal components into static Gaussians to model scene deformations over time. Among them, some approaches adopt isotropic 4D Gaussians [22, 23] or

*Corresponding Author.

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

polynomial functions [24–26] to explicitly represent motion. These representations circumvent dense MLP computation and thus enable efficient rendering.

Despite these success, a key limitation of these methods is the *independent* modeling of motion for each Gaussian. In reality, the dynamics are not independent. They are inherently correlated across space and time, often exhibiting complex, hierarchical structures. Consequently, assuming independence prevents the capture of coherent and structured motion patterns, creating a significant modeling gap. Thus, it is crucial to model deformations collaboratively, allowing Gaussians to share information with their neighbors and better reflect the structured nature of real-world motion.

To achieve this, we introduce **TreeSplat**, a novel representation that employs a tree data structure to explicitly model both individual Gaussian motions and their inter-correlations. The core idea is to organize motion hierarchically within this tree. Specifically, each node stores coefficients for a set of shared, time-varying basis functions, with these coefficients defining a component of the motion. The complete motion for any Gaussian is then determined by aggregating these component transformations—typically via weighted summation—along the path from its associated leaf node to the tree's root. Consequently, the tree explicitly defines the motions for all Gaussians, and crucially, Gaussians that share common ancestor nodes inherently share motion components, thus modeling their correlation.

One central problem is how we construct the tree. Rather than manual or rule-based creation, we adaptively grow the tree in conjunction with Gaussian densification. As standard densification operations, cloning and splitting generate new Gaussians in adjacent regions, the tree structure grows alongside them.

Specifically, we define two forms tree growth modes, that operate alternately. *Leaf Expansion* broadens the tree. It links motion of new Gaussians (resulting from densification) with new leaf nodes under the same parent as their source Gaussian. *Depth Promotion* increases the depth of the tree. It transforms an existing Gaussian's leaf node into a new parent, which then links it to two new child leaf nodes associated with newly generated Gaussians. Furthermore, to control the influence of different tree levels, each Gaussian applies a learnable decay to the coefficients of motion nodes with increasing depth. This ensures that motion components defined by nodes deeper in the tree have a smaller impact, while components closer to the root dictate more significant.

Our motion tree offers a key advantage: its coefficients are **mergeable** after training. This mergeability means that although the tree structure is essential for training, it is not required at inference time. Once training is complete, we can merge the coefficient for each Gaussian by traversing its leaf-to-root path and accumulating the corresponding transformations. Crucially, this merging process occurs *offline*, before test time. Consequently, we eliminate the requirement for per-Gaussian tree traversal and significantly accelerating runtime performance.

To maintain computational efficiency during training, we periodically prune inactive motion nodes, preserving a sparse and lightweight tree structure. Once training converges, the tree structure is fixed, and the motion coefficients are merged offline as described. This final representation, with pre-aggregated motion for each Gaussian, enables highly efficient rendering of complex dynamic scenes without runtime tree traversal overhead.

In summary, the contributions of this work are as follows:

- We propose a hierarchical motion tree structure for dynamic Gaussian Splatting, where the tree is grown in conjunction with Gaussian densification to model collaborative and structured motion.
- The motion tree is formulated as mergeable, in which the linear structure enables offline pre-aggregation of motion coefficients. This design eliminates the need for tree traversal during rendering, thereby substantially improving efficiency and reducing storage overhead.
- Our TreeSplat framework achieves state-of-the-art reconstruction quality on both synthetic and real-world datasets, while maintaining real-time rendering speed over 200 FPS.

2 Related works

Dynamic 3D Scene Reconstruction. 3D Gaussian Splatting (3DGS) [19, 38–40] have become a key approach for efficient 3D scene reconstruction. A stream of recent works [16, 20–36, 41–43] has extended 3DGS to dynamic scene reconstruction by introducing temporal components to model deformations over time. These methods can be broadly divided into two categories based on how they

incorporate temporal components: implicit and explicit deformation modeling. In implicit approaches, several representative methods [20, 21] employ a shared MLP to predict the deformation of each Gaussian over time. This design is similar to deformation components used in dynamic NeRFs. However, real-world dynamic scenes often require millions of Gaussians for accurate reconstruction. As a result, querying an MLP for each Gaussian introduces significant computational overhead, which becomes a major bottleneck during rendering. To mitigate this issue, more recent work has focused on explicit deformation representations that avoid per-Gaussian MLP inference. Such methods either encode deformation using isotropic 4D Gaussians [22, 23] or parameterize the deformation of each Gaussian using polynomial functions of time [24–26]. While these techniques improve efficiency in both training and rendering, they learn motion parameters independently for each Gaussian and ignore the spatial dependencies between motions of Gaussians. This independence assumption neglects the hierarchical and coherent motion patterns often present in dynamic scenes, leaving a critical modeling gap. In contrast, our approach introduces a hierarchical and mergeable motion tree that explicitly models collaborative motion across Gaussians, enabling improved dynamic scene reconstruction and rendering effiency.

Hierarchy in Gaussian Splatting. Hierarchical representations are fundamental in 3D reconstruction. Early works [44, 45] introduced strategies such as spatial partitioning and pyramid features into NeRF [1] to improve both rendering speed and visual quality across diverse 3D scenes. Thanks to its fully explicit formulation, Gaussian Splatting allows for more direct modeling over spatial structure, making it a natural candidate for hierarchical representation. Several recent works [46–50] have explored constructing hierarchical representations based on Gaussian Splatting. HierarchicalGS [46] proposed a tree-based hierarchy that enables the reconstruction of large-scale scenes by dividing them into independent spatial chunks. OctreeGS [48] employed an octree data structure to adaptively capture level-of-detail variations across different scene regions, while SVRaster [47] also adopted an octree structure and introduced a sparse voxel rasterization method to mitigate popping artifacts [51] in Gaussian Splatting rendering.

More recently, HiCoM [49] explored hierarchical motion modeling using 4DGaussian [20] for streamable dynamic scene reconstruction. However, its hierarchy is constructed using fixed heuristics, which limits adaptability across varying scenes. Concurrent work HiMoR [50] introduced a fixed-structure tree of depth 3 to represent motion for monocular dynamic scene reconstruction. Different from these approaches, our framework focuses on mining motion hierarchies between Gaussians in an adaptive manner. We couple tree construction with the densification process, enabling the tree to grow adaptively in both depth and width to reflect motion complexity. Moreover, our linear tree structure supports pre-aggregation of motion coefficients after training, enabling highly efficient rendering without additional tree traversal overhead.

3 Preliminary: 3D Gaussian Splatting

Given a set of images captured from 3D scenes with known camera poses, 3D Gaussian Splatting (3DGS) [19] iteratively reconstructs the scene as a collection of isotropic Gaussians, denoted as $\{\mathcal{G}\}$. Each Gaussian is defined by its central position $\boldsymbol{\mu} \in \mathbb{R}^3$, opacity $\sigma \in [0, 1]$, and view-dependent color $\boldsymbol{h} \in \mathbb{R}^{48}$ represented as spherical harmonics coefficients. Its 3D covariance matrix $\Sigma^{3D} \in \mathbb{R}^{3\times3}$ is parameterized by a scale vector $\boldsymbol{s} \in \mathbb{R}^3$ and a rotation quaternion $\boldsymbol{q} \in \mathbb{R}^4$ to ensure positive semi-definiteness. Collectively, the i-th Gaussian is denoted as $\mathcal{G}_i = \{\boldsymbol{\mu}_i, \sigma_i, \boldsymbol{h}_i, \boldsymbol{s}_i, \boldsymbol{q}_i\}$.

To better represent the scene, adaptive density control, or densification, is employed by dynamically adjusting the number of Gaussians during optimization. It selectively densifies regions with large view-space positional gradients, which typically indicate either missing geometry (under-reconstruction) or excessive coverage (over-reconstruction). Specifically, small Gaussians in under-reconstructed regions are cloned to cover new geometry, while large Gaussians in high-variance areas are split into smaller Gaussians to capture finer scene details.

For rendering a 2D image, all 3D Gaussians are first projected onto the image plane as 2D Gaussians. The opacity of Gaussian i at a pixel $\mathbf{x}$ is given by:

$$\alpha = \sigma_i \exp\left(-\frac{1}{2}(\mathbf{x} - \bar{\boldsymbol{\mu}}_i)^T \Sigma_i^{2D^{-1}} (\mathbf{x} - \bar{\boldsymbol{\mu}}_i)\right), \tag{1}$$

where $\bar{\boldsymbol{\mu}}_i$ and Σ_i^{2D} are the projected mean and covariance, respectively. After sorting Gaussians based on depth, a tile-based rasterizer is used to blend their contributions, producing the final image.

4 Methodology

In this section, we first establish the base formulation of dynamic Gaussian Splatting in Sec. 4.1. The detailed data structure of the motion tree is elaborated in Sec. 4.2, and the tree merging strategy for rendering is described in Sec. 4.3. Finally, the initialization and optimization scheme for dynamic Gaussian Splatting with the motion tree is presented in Sec. 4.4.

4.1 Dynamic Gaussian Splatting

To adapt 3D Gaussian Splatting for dynamic scene reconstruction, we adopt explicit temporal components to capture deformations over time. Specifically, per-Gaussian coefficients associated with shared basis function are applied to encode temporal motion and rotation of each Gaussian as shown in Fig. 1(a). Rather than relying on predefined motion bases such as Fourier series [24] or polynomial [26] basis, we initially represent the motion field through per-Gaussian coefficients [25] with learnable bases:

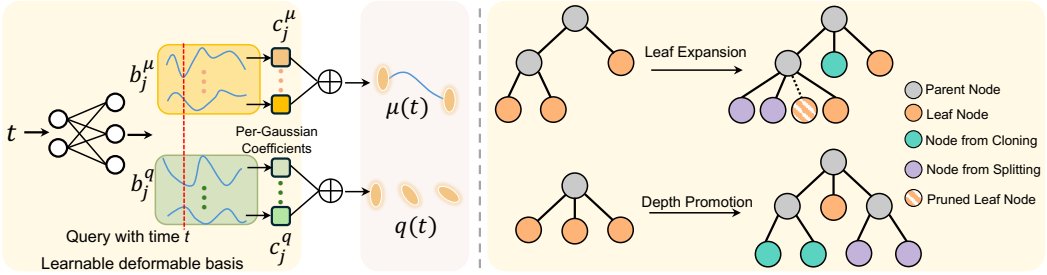
$$\boldsymbol{\mu}(t) = \boldsymbol{\mu}_c + \hat{\boldsymbol{\mu}}(t) = \boldsymbol{\mu}_c + \sum_{j=1}^{B} c_j^{\mu} \boldsymbol{b}_j^{\mu}(t), \quad \boldsymbol{q}(t) = \boldsymbol{q}_c + \hat{\boldsymbol{q}}(t) = \boldsymbol{q}_c + \sum_{j=1}^{B} c_j^{q} \boldsymbol{b}_j^{q}(t), \tag{2}$$

where $\boldsymbol{\mu}_c$ and $\boldsymbol{q}_c$ are the static Gaussian center and rotation quaternion, respectively, and B denotes the number of basis functions. Here, $c_j^{\mu} \in \mathbb{R}$ and $c_j^{q} \in \mathbb{R}$ are learnable coefficients. The basis functions $\boldsymbol{b}_j^{\mu}(t) \in \mathbb{R}^3$ and $\boldsymbol{b}_j^{q}(t) \in \mathbb{R}^4$ are shared across all Gaussians and are generated by a lightweight multi-layer perceptron (MLP) f_θ with a sinusoidal time embedding:

$$f_\theta(\gamma(t)) = \{(\boldsymbol{b}_j^{\mu}(t), \boldsymbol{b}_j^{q}(t))\}_{j=1}^{B}, \quad \gamma(t) = \left(\sin(2^k \pi t), \cos(2^k \pi t)\right)_{k=0}^{L-1}, \tag{3}$$

where θ denotes the learnable parameters of the MLP, and $\gamma(t)$ is a sinusoidal positional encoding of order L to enhance the network's capacity to model high-frequency temporal variations.

Compared to MLP-based deformation modeling that requires querying the MLP millions of times [20, 21], once for each Gaussian, our framework only performs a single query to f_θ per timestamp t, substantially reducing the computational overhead.



(a) Dynamic Gaussian with Learnable Basis (b) Leaf Expansion and Depth Promotion

Figure 1: **Dynamic Gaussian Splatting with Hierarchical Motion Tree.** Our framework encodes temporal motion and rotation using shared learnable basis functions combined with per-Gaussian coefficients. Each node in the hierarchical motion tree stores a motion vector parameterized by learnable coefficients. The tree is constructed in conjunction with Gaussian densification, where *Leaf Expansion* and *Depth Promotion* are performed periodically to adaptively grow the hierarchy.

4.2 Hierarchical Motion Tree

Though the above dynamic Gaussian Splatting models per-Gaussian deformation using shared basis functions, the associated coefficients are optimized independently for each Gaussian. However, motions in real-world 3D scenes are inherently spatially entangled and exhibit hierarchical relationships across multiple scales, where the motion of one structure is often correlated with the motions of its neighbors. Motivated by this property, we introduce a hierarchical motion tree to collaboratively learn the temporal motion $\hat{\boldsymbol{\mu}}(t)$ in Eq. 2 among Gaussians.

In the motion tree structure, each node represents a motion vector encoded as a learnable coefficient vector in $\mathbb{R}^B$, and the entire motion field is denoted as $\mathcal{F} \in \mathbb{R}^{M \times B}$, where M is the total number of

nodes in the tree. Each leaf node is uniquely associated with one Gaussian, ensuring a one-to-one correspondence between Gaussians and leaf nodes. Given N ($N \leq M$) dynamic Gaussians, the structure of the motion tree is maintained by the following data mappings:

- Node entry $\mathcal{E} \in \mathbb{N}^N$: maps each Gaussian to its leaf node index in the motion field $\mathcal{F}$;
- Parent index $\mathcal{P} \in \mathbb{N}^M$: records the parent motion node for each motion node;
- Node level $\mathscr{L} \in \mathbb{N}^N$: specifies the number of nodes from Gaussian's leaf node to the root node;

Given a constructed motion tree, the motion $\hat{\boldsymbol{\mu}}_i(t)$ of Gaussian i is computed by traversing its ancestral path with parent index $\mathcal{P}$ and aggregating motion vectors. Formally, we can formulate as:

$$\hat{\boldsymbol{\mu}}_i(t) = \sum_{k=0}^{\ell_i} \sum_{j=1}^{B} \mathcal{F}_j^{\mathrm{anc}(i,k)} \boldsymbol{b}_j^{\mu}(t), \tag{4}$$

where ℓ_i denotes the level of Gaussian i in the tree, and k indexes the nodes along its path from the leaf to the root. The function $\mathrm{anc}(i,k)$ returns the index in $\mathcal{F}$ of the ancestor node k steps above the leaf node of Gaussian i.

Tree Growth: Leaf Expansion and Depth Promotion. As Gaussian densification progressively refines the scene from coarse to fine spatial levels by creating new Gaussians in neighboring regions, it naturally aligns with the hierarchical motion modeling motivation. The growth of the motion tree is in conjunction with the densification during optimization. To adaptively balance the depth and width of the motion tree, we introduce two alternative modes to grow the tree at each densification step as shown in Fig. 1(b).

Leaf Expansion increases the number of leaf nodes horizontally without increasing tree depth. Newly densified Gaussians are linked to new leaf nodes under the same parent node as their source Gaussian. These new leaf nodes share the same parent index $p \in \mathbb{N}$ and node level $l \in \mathbb{N}$ as the leaf node of their source Gaussian, preserving the flatness of the local tree structure.

Depth Promotion deepens the tree to refine the hierarchical organization of motions. It transforms an existing Gaussian's leaf node into a new parent node. The densification operation, whether cloning or splitting, is treated as a binary operation that creates two new leaf nodes associated with Gaussians and connect them to the newly promoted parent node. These new leaf nodes are assigned a node level of $l + 1$ and their parent indices p point to the newly promoted parent node.

For all newly created leaf nodes, the associated coefficients in the motion field $\mathcal{F}$ are initialized to zero. This initialization ensures that the aggregated temporal motion $\hat{\boldsymbol{\mu}}(t)$ remains unchanged immediately before and after densification.

This two growth modes are scheduled periodically during optimization: multiple rounds of Leaf Expansion are followed by a Depth Promotion step. This design results in a large, adaptively widening motion tree with limited depth, ensuring that the tree remains both scalable and capable of capturing multi-level motion structures throughout training. Besides, unused leaf nodes are periodically remove during optimization to maintain efficiency during training.

Decay-Weighted Aggregation. As the motion tree deepens, naive aggregation of ancestor nodes, as formulated in Eq. 4, can result in unstable optimization. Specifically, given the upstream per-Gaussian gradient $\boldsymbol{g}_i = \partial \mathcal{L} / \partial \hat{\boldsymbol{\mu}}_i$, the gradient with respect to an ancestor node $\mathcal{F}^{\mathrm{anc}(i,k)}$ is computed as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{F}^{\mathrm{anc}(i,k)}} = \left(\boldsymbol{g}_i^\top \boldsymbol{b}_1^{\mu}(t),\, \boldsymbol{g}_i^\top \boldsymbol{b}_2^{\mu}(t),\, \ldots,\, \boldsymbol{g}_i^\top \boldsymbol{b}_B^{\mu}(t) \right) \in \mathbb{R}^B \tag{5}$$

In this naive aggregation, ancestor node accumulates the full strength of the upstream gradient from each of its descendants. Consequently, the total gradient magnitude at a node grows proportionally to the number of its descendant Gaussians, which can cause severe gradient explosion for some ancestor nodes and destabilize optimization.

To mitigate this issue, we introduce a learnable decay factor $\beta_i \in (0, 1]$ for each Gaussian. The motion aggregation is reformulated as:

$$\hat{\boldsymbol{\mu}}_i(t) = \sum_{k=0}^{\ell_i} \left(\prod_{p=0}^{k-1} \beta_i \right) \sum_{j=1}^{B} \mathcal{F}_j^{\mathrm{anc}(i,k)} \boldsymbol{b}_j^{\mu}(t), \tag{6}$$

where deeper ancestors are exponentially attenuated by powers of β_i. Correspondingly, gradients received by each ancestor node is scaled by the same decay factor, ensuring that gradients from distant descendants are substantially suppressed. The decay factors β_i are differentiable and jointly optimized with other model parameters during training. A closed-form derivation of their gradients is provided in the *Appendix*.

This *decay-weighted aggregation* strategy enables each Gaussian to adaptively control its receptive field within the motion tree: a smaller β_i concentrates motion learning on nearby ancestors, while a larger β_i permits broader aggregation. By suppressing contributions from distant and irrelevant ancestors, it improves the robustness of both forward motion modeling and backward gradient flow, leading to more stable motion modeling.

4.3 Tree Merging for Rendering

During optimization, computing the motion $\hat{\boldsymbol{\mu}}_i(t)$ for each Gaussian requires traversing its ancestral path in the deformation tree. Once training is complete and the tree structure is fixed, we pre-aggregate the motion coefficients for each Gaussian by reordering the summation terms in Eq. 6. Formally, it can be expressed as:

$$\hat{\boldsymbol{\mu}}_i(t) = \sum_{j=1}^{B} \boldsymbol{b}_j^{\mu}(t) \sum_{k=0}^{\ell_i} \left(\prod_{p=0}^{k-1} \beta_i \right) \mathcal{F}_j^{\mathrm{anc}(i,k)} = \sum_{j=1}^{B} \boldsymbol{b}_j^{\mu}(t) \boldsymbol{c}_j^i \tag{7}$$

where $\boldsymbol{c}^i \in \mathbb{R}^B$ is the merged motion coefficient vector for Gaussian i. This transformation enables highly efficient rendering, as it eliminates the need to traverse the tree, the motion computation reduces to simple linear combination over the B basis vectors per Gaussian.

It is important to note that this coefficient merging is only applied after training. During optimization, the internal motion nodes must be retained to receive gradient signals from multiple descendant Gaussians and propagate them back across the tree. Merging would prematurely remove the hierarchical structure necessary for effective gradient flow and motion disentanglement. Hence, mergeability is a property leveraged only in the inference stage to accelerate rendering without compromising the training dynamics.

4.4 Training Framework

Following 3D Gaussian Splatting [19], we conduct interleaved iterative optimization and densification. Since the tree construction process is coupled with densification, the time overhead introduced by growing the deformation tree is negligible. Moreover, with our highly optimized CUDA implementation, tree traversal for each Gaussian during optimization introduces only a marginal time increase.

Tree Initialization. We first use point cloud to initialize N_{init} Gaussians. The node levels $\mathscr{L}$ are initialized to 0 for all Gaussians, corresponding to the root level of the tree. The initial motion field $\mathcal{F}$ has the same length as the number of Gaussians, with all entries initialized to zero. For the node entry mapping $\mathcal{E}$, each Gaussian is assigned to its corresponding motion node in order, initialized as $[0, 1, \ldots, N_{\text{init}} - 1]$. The parent indices $\mathcal{P}$ are initialized to -1 for all nodes, forming a forest of independent singleton trees at initialization.

Optimization Scheme. During training, we simply adopt a rendering loss composed of an MSE term and an SSIM term: $\mathcal{L} = (1 - \lambda)\mathcal{L}_{\text{mse}} + \lambda\mathcal{L}_{\text{ssim}}$, where λ balances the two objectives. To stabilize training, we initially reconstruct the static components of $\boldsymbol{\mu}(t)$ and $\boldsymbol{q}(t)$ over the first 10% of optimization iterations, effectively learning a canonical 3D space before introducing temporal dynamics. For interleaved densification, we adopt the averaged screen-space 2D gradient of $\boldsymbol{\mu}(t)$ as the density control indicator.

5 Experiments

5.1 Implementation details

To balance efficiency and performance, we model the time-varying shared basis f_θ in Eq. 3 using a MLP with three hidden layers, each of width 512. The sinusoidal time embedding $\gamma(t)$ is set

as order $L = 32$ to capture high-frequency temporal variations. The number of basis vectors B is set to 10 for the D-NeRF dataset and 16 for the Neural 3D Video dataset, to accommodate the complexity of real-world scenes. Motion tree construction begins at iteration 500 in conjunction with Gaussian densification. *Depth Promotion* is performed at first step, after which *Leaf Expansion* is applied every 100 iterations and *Depth Promotion* every 500 iterations, alternating after every four rounds of *Leaf Expansion*. Densification is halted at iteration 15,000, after which the motion tree is fixed to capture stable collaborative motion patterns among Gaussians. To ensure fast per-frame motion computation, we implement Eq. 6 and Eq. 7 as custom CUDA kernels separately. Through careful kernel design, including memory access optimization and parallel thread scheduling, we ensure that tree traversal overhead is negligible even for real-world dynamic scenes. All other training hyperparameters, including densification thresholds and learning rates for Gaussian attributes, follow the original settings used in Gaussian Splatting. The learning rate for the decay factor β is set to 5×10^{-3}, and the learning rate for the shared MLP f_θ is set to 5×10^{-4}. All experiments are conducted on an NVIDIA RTX4500 Ada GPU. For additional hyperparameter configurations, please refer to the configuration table provided in the *Appendix*.

5.2 Comparison with State-of-the-art

D-NeRF Dataset [7]. The D-NeRF dataset comprises 8 synthetic dynamic scenes captured as monocular videos. At each time step, only a single training image from one viewpoint is available. Following standard protocols, we evaluate on test views from novel camera positions within the same temporal range as the training data. For initialization, we uniformly sample 100,000 points within the cubic volume $[-1.2, 1.2]^3$. Quantitative results are reported in Tab. 1 in terms of PSNR, SSIM, and LPIPS. Our TreeSplat achieves the highest reconstruction quality with an average PSNR of 37.11 dB, while maintaining a compact model size of 28 MB and requiring only 4 minutes of training per scene. After tree merging, TreeSplat reaches a rendering speed of 230 FPS, outperforming all baselines. Moreover, it uses significantly fewer Gaussians, averaging only 85K per scene, demonstrating superior efficiency without compromising quality.

Table 1: Quantitative Evaluation on the D-NeRF [7] Dataset.

Method	PSNR↑	SSIM↑	LPIPS↓	Train Time↓	FPS↑	Storage↓	#Gauss↓
DNeRF [7]	29.67	0.95	0.08	40 h	0.1	N/A	N/A
K-Planes [52]	31.07	0.97	0.02	5 h	1.2	N/A	N/A
HexPlanes [8]	31.04	0.97	0.04	11 min	0.22	N/A	N/A
TiNeuVox [53]	32.67	0.97	0.04	49 min	1.6	N/A	N/A
4DGaussian [20]	34.05	0.98	0.02	19 min	85	19M	137K
CompactDGS [24]	32.19	0.97	0.04	4 min	202	32M	112K
DynMF [25]	35.72	0.98	0.02	5 min	216	32M	101K
4DRotorGS [22]	34.26	0.97	0.03	26 min	143	242M	392K
RealTime4DGS [23]	34.09	0.98	0.02	28 min	132	278M	445K
TreeSplat (ours)	37.11	0.98	0.02	4 min	230	28M	85K

Neural 3D Video Dataset. The Neural 3D Video (N3V) dataset consists of 6 indoor dynamic scenes captured with 18 to 21 cameras at a resolution of 2704×2028, each lasting ten seconds. Following standard protocols, we train and evaluate at the half resolution, using 300 frames per scene with the center camera held out for evaluation. For initialization, we sample 300,000 points from the COLMAP [54] point cloud using farthest point sampling [55]. Tab. 2 reports quantitative results in terms of PSNR, storage size, rendering speed, and training time. We compare our method against 7 NeRF-based approaches and 5 representative Gaussian Splatting methods. Among them, 4DGaussian [20] and Grid4D [35] employ MLP-based modeling, while DynMF [25], RealTime4DGS [23], and SpaceTimeGS [26] adopt explicit deformation representations. Our TreeSplat achieves the highest average reconstruction quality of 32.21 dB across scenes, while maintaining a compact model size of 170 MB and the fastest rendering speed of 206 FPS after tree merging. Remarkably, TreeSplat also trains significantly faster than most baselines, requiring only 0.57 hours on average per scene. Besides, we qualitatively compare novel view synthesis results on two representative scenes (*Coffee Martini* and *Sear Steak*) in Fig. 2, showing our method yields better reconstruction. These results highlight TreeSplat's superior trade-off between reconstruction quality and efficiency.

Table 2: Quantitative Evaluation on the Neural 3D Video [12] Dataset.

Method	PSNR (dB)							MB	Frame/s	Hours
	Coffee Martini	Cook Spinach	Cut Roasted Beef	Flame Salmon	Flame Steak	Sear Steak	**Average**	Size	FPS	Training time
NeRFPlayer [13]	31.53	30.56	29.35	31.65	31.93	29.13	30.69	5130	0.05	6
HyperReel [14]	28.37	32.30	32.92	28.26	32.20	32.57	31.10	360	2	9
DyNeRF [12]	N/A	N/A	N/A	29.58	N/A	N/A	29.58	28	0.015	1344
HexPlane [8]	N/A	32.04	32.55	29.47	32.08	32.39	31.71	200	N/A	12
K-Planes [8]	29.99	32.60	31.82	30.44	32.38	32.52	31.63	311	0.3	1.8
MixVoxels-L [15]	29.63	32.25	32.40	29.81	31.83	32.10	31.34	500	37.7	1.3
MixVoxels-X [15]	30.39	32.31	32.63	30.60	32.10	32.33	31.73	500	4.6	N/A
4DGaussian [20]	27.34	32.46	32.90	29.20	32.51	32.49	31.15	34	137	1.7
Grid4D [35]	28.30	32.58	33.22	29.12	32.56	33.16	31.49	146	116	1.9
DynMF [25]	28.87	33.09	32.66	29.03	32.70	32.02	31.40	176	197	0.52
RealTime4DGS [23]	28.33	32.93	33.85	29.38	34.03	33.51	32.01	2085	101	5.2
SpaceTimeGS [26]	28.61	33.18	33.52	29.48	33.64	33.89	32.05	200	140	5.5
TreeSplat (ours)	28.91	33.17	33.69	29.50	33.89	34.10	32.21	170	206	0.57

Figure 2: **Qualitative comparisons on the Neural 3D Video Dataset.** We qualitatively compare our TreeSplat with 4DGaussian [20], DynMF [25], and RealTime4DGS [23] on two representative scenes *coffee martini* and *sear steak* from the Neural 3D Video dataset [12].

5.3 Ablation and Discussion

Effects of Tree Depth. Our hierarchical motion tree grows in conjunction with Gaussian densification through periodically scheduled *Leaf Expansion* and *Depth Promotion*. The overall tree depth is controlled by adjusting the interval between *Depth Promotion* steps. In Tab. 3, we conduct an ablation study across five configurations with maximum tree depths of 0, 4, 15, 29, and 73, corresponding to *Depth Promotion* intervals of ∞, 4000, 1000, 500, and 200, respectively. The results show that performance consistently improves as tree depth increases. Compared to the baseline without a hierarchical structure (tree depth 0), our full model with depth 29 achieves notable PSNR gains, such as $+2.63$ on *Hellwarrior* and $+1.40$dB on *Sear Steak*. Even moderate depths (e.g., 4 or 15) already yield substantial improvements, demonstrating that our tree structure captures meaningful motion correlations with limited hierarchy.

Table 3: **Ablation of tree depth.** Evaluated on representative scenes from D-NeRF (*Hellwarrior*, *Mutant*) and Neural 3D Video (*Cut Roasted Beef*, *Sear Steak*).

Tree Depth	**Hellwarrior**		**Mutant**		**Cut Roasted Beef**		**Sear Steak**	
	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
73	38.26	0.97	43.28	1.00	33.31	0.96	34.05	0.97
29	38.34	0.97	43.81	1.00	33.69	0.96	34.10	0.97
15	38.11	0.97	42.57	1.00	32.79	0.96	33.81	0.97
4	37.49	0.97	42.79	1.00	33.63	0.96	33.39	0.96
0	35.71	0.95	41.47	0.99	32.15	0.96	32.65	0.96

However, excessively deep trees can reduce the width of the tree and may accumulate irrelevant motion noise. Moreover, they introduce greater computational overhead and may even cause minor performance degradation. Empirically, we find that a moderate depth of 29 (interval 500) offers the best balance between performance, efficiency, and robustness.

Effects of Tree Merging. The main difference between the training-time motion computation in Eq. 6 and the rendering-time formulation in Eq. 7 lies in the order of aggregation. During training, we first compute time-dependent motion vectors for all nodes in the motion tree and then aggregate them for each Gaussian. In contrast, rendering pre-aggregates the motion coefficients per Gaussian before applying the basis functions, thereby eliminating the need to traverse the tree. While these two formulations are mathematically equivalent, numerical differences can arise due to floating-point accumulation order. To evaluate the impact of such differences, we conduct ablation experiments comparing models with and without coefficient merging. As shown in Tab. 4, merging introduces negligible variation in reconstruction quality across PSNR, SSIM, and LPIPS, while significantly improving rendering speed and reducing storage size. These results validate the effectiveness of our mergeable design, which preserves visual fidelity while enabling high-performance inference.

Table 4: **Ablation on Tree Merging.** We report the metric difference as "after merging" minus "before merging." Merging preserves reconstruction quality while improving efficiency and compactness.

Scene	ΔPSNR	ΔSSIM	ΔLPIPS	ΔStorage (MB)	ΔFPS
Hellwarrior	-3.81×10^{-7}	$+5.96 \times 10^{-9}$	-4.47×10^{-8}	-0.62	$+15$
Mutant	$+9.53 \times 10^{-7}$	$+2.98 \times 10^{-9}$	-1.97×10^{-8}	-1.55	$+12$
Cut Roasted Beef	$+3.81 \times 10^{-8}$	-7.94×10^{-10}	-7.45×10^{-9}	-8.57	$+82$
Sear Steak	-1.27×10^{-8}	$+1.39 \times 10^{-9}$	$+7.45 \times 10^{-9}$	-8.61	$+74$

Effects of Weight-Decayed Aggregation. To assess the impact of weight-decayed aggregation, we conduct ablation by disabling it during training on the D-NeRF dataset. As illustrated in Fig. 3, removing this strategy leads to blurred motion regions in the rendered results. This is caused by interference from irrelevant nodes in the motion tree. In contrast, enabling this strategy yields much better reconstruction quality, as the learnable decay factor β in Eq. 6 adaptively controls each Gaussian receptive field in the motion tree.

Scene	w/o Window		With Window	
	PSNR	#Gauss	PSNR	#Gauss
Hellwarrior	38.34	19K	37.87	22K
Mutant	43.81	70K	41.35	79K
Cut roasted beef	33.69	416K	33.36	460K
Sear steak	34.10	429K	33.89	519K

Table 5: **Impact of Temporal Opacity window.** We compare with and without opacity window.

Figure 3: **Ablation of weight-decayed aggregation.**

Analysis of Tree Depths. To better understand the structure of motion hierarchies, we analyze the distribution of tree depths across four representative scenes: *Hellwarrior* and *Mutant* from the DNeRF dataset, and *Cut Roasted Beef* and *Sear Steak* from the Neural 3D Video dataset. As shown in Figure 4, we divide tree depths into six intervals and visualize the results as histograms. For each interval, we also report the average number of Gaussians per tree. The resulting distributions exhibit a long-tailed pattern: over 97% of motion trees have depths less than or equal to 10, while only a small fraction extend into deeper hierarchies. These findings demonstrate that most Gaussians are captured through relatively shallow hierarchical motion patterns, with deeper structures emerging only where necessary.

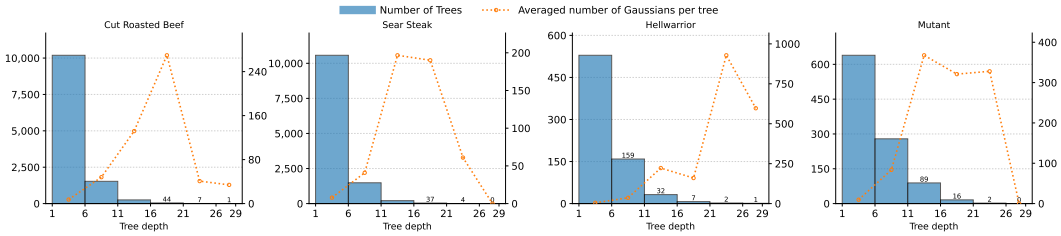


Figure 4: **Visualization of hierarchical tree statistics.** Here we show statistics across four dynamic 3D scenes (*Cut Roasted Beef*, *Sear Steak*, *Hellwarrior*, and *Mutant*). For each scene, the histogram (blue bars) represents the number of trees at different depth intervals, while the dotted orange curve denotes the averaged number of Gaussians per tree within each depth range.

Temporal Opacity. Some prior works [23, 26] model the opacity σ of each Gaussian as a time-varying unimodal function, often implemented as a temporal window. However, recent findings [56, 57] suggest that Gaussians tend to learn overly short temporal windows, resulting in limited lifespans and degrading dynamic 3D reconstruction into near per-frame static modeling. To evaluate the necessity of temporal opacity in our framework, we follow the same design and assign each Gaussian a learnable 1D Gaussian window. As reported in Tab. 5, introducing temporal windows degrades reconstruction quality despite a noticeable increase in the number of Gaussians. This likely occurs because Gaussians with limited temporal visibility hinder the motion tree from learning coherent motion trajectories. These results confirm that temporal windowing is unnecessary in our framework. The temporal modulation of opacity can be sufficiently expressed through Gaussian center motion, as reflected in the spatial formulation of opacity in Eq. 1.

6 Conclusion

In this work, we present a hierarchical and mergeable motion tree structure for dynamic Gaussian Splatting. By constructing motion tree in conjunction with Gaussian densification, our method enables collaborative motion learning across Gaussians in an adaptive, coarse-to-fine manner. A decay-weighted aggregation strategy further regulates the influence of ancestor nodes, improving both optimization stability and motion locality. To support efficient rendering, we introduced a pre-aggregation strategy that merges motion coefficients over the tree path of each Gaussian, eliminating traversal overhead during rendering. Extensive experiments on both synthetic and real-world datasets demonstrate that our method achieves state-of-the-art reconstruction quality while maintaining high efficiency and compactness. Our approach offers a principled framework for hierarchical motion modeling and sets the stage for future extensions in scalable dynamic 3D representations.

Acknowledgement

This project is supported by the National Research Foundation, Singapore, under its Medium Sized Center for Advanced Robotics Technology Innovation.

References

[1] 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. In *ECCV*, 2020.

[2] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 2022.

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022.

[4] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2856–2865, 2021.

[5] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5501–5510, 2022.

[6] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5741–5751, 2021.

[7] 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.

[8] Ang Cao and Justin 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.

[9] 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. *Advances in Neural Information Processing Systems*, 35:36762–36775, 2022.

[10] Yu-Lun Liu, Chen Gao, Andreas Meuleman, Hung-Yu Tseng, Ayush Saraf, Changil Kim, Yung-Yu Chuang, Johannes Kopf, and Jia-Bin Huang. Robust dynamic radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13–23, 2023.

[11] Sungheon Park, Minjung Son, Seokhwan Jang, Young Chun Ahn, Ji-Yeon Kim, and Nahyup Kang. Temporal interpolation is all you need for dynamic neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4212–4221, 2023.

[12] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5521–5531, 2022.

[13] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. Nerfplayer: A streamable dynamic scene representation with decomposed neural radiance fields. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2732–2742, 2023.

[14] Benjamin Attal, Jia-Bin Huang, Christian Richardt, Michael Zollhoefer, Johannes Kopf, Matthew O'Toole, and Changil Kim. Hyperreel: High-fidelity 6-dof video with ray-conditioned sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16610–16620, 2023.

[15] Feng Wang, Sinan Tan, Xinghang Li, Zeyue Tian, Yafei Song, and Huaping Liu. Mixed neural voxels for fast multi-view video synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19706–19716, 2023.

[16] Zhen Xu, Sida Peng, Haotong Lin, Guangzhao He, Jiaming Sun, Yujun Shen, Hujun Bao, and Xiaowei Zhou. 4k4d: Real-time 4d view synthesis at 4k resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20029–20040, 2024.

[17] Haotong Lin, Sida Peng, Zhen Xu, Tao Xie, Xingyi He, Hujun Bao, and Xiaowei Zhou. High-fidelity and real-time novel view synthesis for dynamic scenes. In *SIGGRAPH Asia*, 2023.

[18] Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. Tensor4d: Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16632–16642, 2023.

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

[20] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[21] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[22] Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 4d-Rotor Gaussian Splatting: Towards Efficient Novel View Synthesis for Dynamic Scenes. In *ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia (SIGGRAPH Asia)*, page 87, 2024.

[23] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time Photorealistic Dynamic Scene Representation and Rendering with 4d Gaussian Splatting. In *The Twelfth International Conference on Learning Representations*, 2024.

[24] 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 (ECCV)*, pages 394–412, 2025.

[25] 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.

[26] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[27] Deqi Li, Shi-Sheng Huang, Zhiyuan Lu, Xinran Duan, and Hua Huang. St-4dgs: Spatial-Temporally Consistent 4d Gaussian Splatting for Efficient Dynamic Scene Rendering. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24*, pages 1–11. ACM, 2024.

[28] Hanyang Kong, Xingyi Yang, and Xinchao Wang. Efficient Gaussian Splatting for Monocular Dynamic Scene Rendering via Sparse Time-Variant Attribute Modeling. In *AAAI Conference on Artificial Intelligence*, volume abs/2502.20378, 2025.

[29] Zhicheng Lu, Xiang Guo, Le Hui, Tianrui Chen, Min Yang, Xiao Tang, Feng Zhu, and Yuchao Dai. 3d geometry-aware deformable gaussian splatting for dynamic view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8900–8910, 2024.

[30] Ruijie Zhu, Yanzhe Liang, Hanzhi Chang, Jiacheng Deng, Jiahao Lu, Wenfei Yang, Tianzhu Zhang, and Yongdong Zhang. Motiongs: Exploring explicit motion guidance for deformable 3d gaussian splatting. *Advances in Neural Information Processing Systems*, 37:101790–101817, 2024.

[31] 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.

[32] Yihua Huang, Yangtian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Sc-GS: Sparse-Controlled Gaussian Splatting for Editable Dynamic Scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[33] Junoh Lee, ChangYeon Won, Hyunjun Jung, Inhwan Bae, and Hae-Gon Jeon. Fully explicit dynamic gaussian splatting. *Advances in Neural Information Processing Systems*, 37:5384–5409, 2024.

[34] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *2024 International Conference on 3D Vision (3DV)*, pages 800–809. IEEE, 2024.

[35] Jiawei Xu, Zexin Fan, Jian Yang, and Jin Xie. Grid4d: 4d decomposed hash encoding for high-fidelity dynamic gaussian splatting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[36] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian splatting for static and dynamic radiance fields. *arXiv preprint arXiv:2408.03822*, 2024.

[37] Youtian Lin, Zuozhuo Dai, Siyu Zhu, and Yao 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.

[38] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19447–19456, 2024.

[39] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20654–20664, 2024.

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

[41] Runze Fan, Jian Wu, Xuehuai Shi, Lizhi Zhao, Qixiang Ma, and Lili Wang. Fov-gs: Foveated 3d gaussian splatting for dynamic scenes. *IEEE Transactions on Visualization and Computer Graphics*, 2025.

[42] Jinbo Yan, Rui Peng, Luyang Tang, and Ronggang Wang. 4d gaussian splatting with scale-aware residual field and adaptive optimization for real-time rendering of temporally complex dynamic scenes. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 7871–7880, 2024.

[43] Jinbo Yan, Rui Peng, Zhiyan Wang, Luyang Tang, Jiayu Yang, Jie Liang, Jiahao Wu, and Ronggang Wang. Instant gaussian stream: Fast and generalizable streaming of dynamic scene reconstruction via gaussian splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 16520–16531, 2025.

[44] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 14335–14345, 2021.

[45] Haithem Turki, Michael Zollhöfer, Christian Richardt, and Deva Ramanan. Pynerf: Pyramidal neural radiance fields. *Advances in neural information processing systems*, 36:37670–37681, 2023.

[46] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics (TOG)*, 43(4):1–15, 2024.

[47] Cheng Sun, Jaesung Choe, Charles Loop, Wei-Chiu Ma, and Yu-Chiang Frank Wang. Sparse voxels rasterization: Real-time high-fidelity radiance field rendering. *ArXiv*, abs/2412.04459, 2024.

[48] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *arXiv preprint arXiv:2403.17898*, 2024.

[49] Qiankun Gao, Jiarui Meng, Chengxiang Wen, Jie Chen, and Jian Zhang. Hicom: Hierarchical Coherent Motion for Dynamic Streamable Scenes with 3d Gaussian Splatting. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[50] Yiming Liang, Tianhan Xu, and Yuta Kikuchi. Himor: Monocular deformable gaussian reconstruction with hierarchical motion representation. *CVPR*, 2025.

[51] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering. *ACM Transactions on Graphics (TOG)*, 43(4):1–17, 2024.

[52] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12479–12488, 2023.

[53] 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. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022.

[54] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979.

[55] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[56] Yuheng Yuan, Qiuhong Shen, Xingyi Yang, and Xinchao Wang. 1000+ fps 4d gaussian splatting for dynamic scene rendering. *arXiv preprint arXiv:2503.16422*, 2025.

[57] 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. *arXiv preprint arXiv:2410.13613*, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: TreeSplat achieves improved reconstruction quality both on synthetic and real world dynamic 3D datasets with collaboratively learn motions between Gaussians.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitation of TreeSplat is discussed and included in the *Appendix*.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: For the theoretical tree merging, we give detail mathematical analysis, numerical errors are also evaluated in the ablation study.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have clearly describe our key algorithm and configuration parameters are also elaborated in *Appendix* to ensure reproduction.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: All code and results of the submission will be made public upon acceptance.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Detailed ablation study is provided in the main paper, and hypera parameters configuration Table is included in the *Appendix*.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: All metrics reported in this work is deterministic reconstruction metric.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Running devices are reported in experiments section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, all datasets used in this paper is public available, this work have cited their papers.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper does not release new assets

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The core method development in this research does not involve LLMs.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.