

# 4D Scaffold Gaussian Splatting for Efficient Dynamic Scene Reconstruction

Woong Oh Cho

In Cho

Seoha Kim

Seon Joo Kim

Yonsei University

Jeongmin Bae

Youngjung Uh

{wocho, join, hailey07, jaymin.bae, yj.uh, seonjookim}@yonsei.ac.kr

## Abstract

Existing 4D Gaussian methods for dynamic scene reconstruction offer high visual fidelity and fast rendering. However, these methods suffer from excessive memory and storage demands, which limits their practical deployment. This paper proposes a 4D anchor-based framework that retains visual quality and rendering speed of 4D Gaussians while significantly reducing storage costs. Our method extends 3D scaffolding to 4D space, and leverages sparse 4D grid-aligned anchors with compressed feature vectors. Each anchor models a set of neural 4D Gaussians, each of which represent a local spatiotemporal region. In addition, we introduce a temporal coverage-aware anchor growing strategy to effectively assign additional anchors to under-reconstructed dynamic regions. Our method adjusts the accumulated gradients based on Gaussians' temporal coverage, improving reconstruction quality in dynamic regions. To reduce the number of anchors, we further present enhanced formulations of neural 4D Gaussians. These include the neural velocity, and the temporal opacity derived from a generalized Gaussian distribution. Experimental results demonstrate that our method achieves state-of-the-art visual quality and 97.8% storage reduction over 4DGS.

## 1. Introduction

Reconstructing dynamic scenes from multi-view videos has received significant attention due to its wide applications in augmented reality and virtual reality. Beyond methods based on neural radiance fields (NeRFs) [1, 12, 13, 21, 24, 28], 3D Gaussian Splatting (3DGS) [8] has become a major approach for dynamic scene reconstruction, due to its ability to render high-quality novel views in real-time. The two main categories of this approach are 1) modeling temporal changes as deformations of 3D Gaussians [2, 31, 34] and 2) employing 4D Gaussians to approximate a scene's 4D volumes [14, 35].

Directly optimizing 4D Gaussians offers faster rendering and higher visual quality than deformation-based meth-

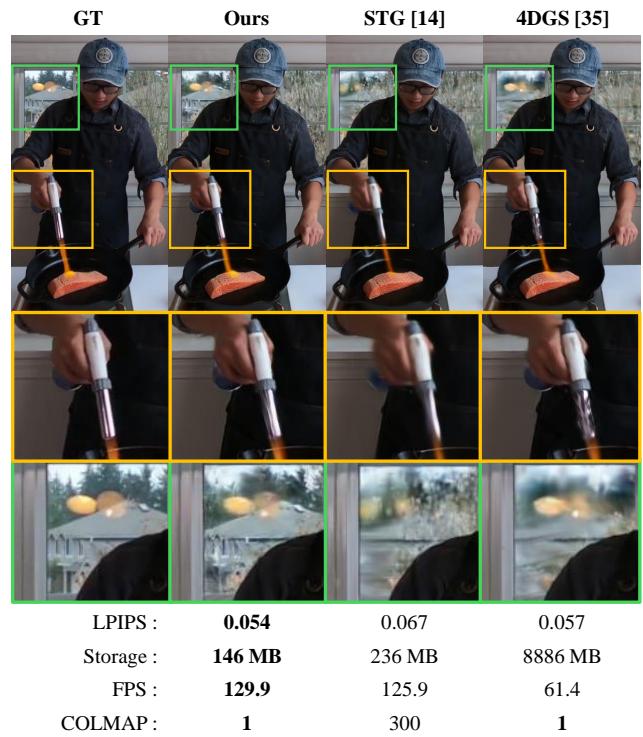


Figure 1. **Teaser.** Our method achieves superior image quality, high rendering speed, and low storage with single-frame COLMAP [25] points, surpassing other 4D Gaussian methods.

ods. Gaussians defined in 4D space can represent sudden changes in the scene, without computing numerous deformations of Gaussians. However, these advantages come at a storage cost: 4D Gaussians demand substantial storage, often exceeding 6GB for 10-second videos. The large number of 4D Gaussians, coupled with extensive parameter counts for 4D properties, hinders scaling this approach for real-world applications. We aim to reduce the high storage demands of 4D Gaussians, by addressing excessively repetitive patterns in dynamic scenes across space and time.

In this paper, we introduce a 4D anchor-based framework for dynamic scene reconstruction, a method that retains the

visual quality and the fast rendering speed of 4D Gaussians with significantly reduced storage. In our method, dynamic scenes are represented through structured anchors aligned with a sparse 4D grid. Each anchor holds a compressed feature vector that models a set of neural 4D Gaussians, each representing a local spatiotemporal region. The properties of neural Gaussians are dynamically generated from the anchor feature vector via shared MLPs.

Our method is a non-trivial extension of 3D scaffolding [17] to 4D space, as it should carefully consider *how to effectively capture dynamic regions with fewer anchors*. Previous 3D scaffolding accumulates gradients of Gaussians to create new anchors, without accounting for temporal coverage. This penalizes dynamic regions that appear shortly, in contrast to background regions that are present in most frames and thus accumulate larger gradients.

We present a temporal coverage-aware anchor growing strategy to allocate new anchors to under-reconstructed dynamic regions. Our approach adjusts the accumulated gradients based on each Gaussian’s temporal coverage: Gaussians with shorter coverage are adjusted to have larger gradients. This modification prevents under-reconstructed dynamic regions from being penalized for their short appearance periods. While the proposed anchor growing operation improves the quality of reconstructed dynamic regions, it also leads to a substantial increase in the number of anchors, and associated storage costs. Such increase originates from the limited capability of generated 4D Gaussians to sufficiently capture real-world dynamics.

To this end, we propose two formulations that enhance our neural Gaussians to better represent actual scene elements. First, we model the time-varying positions of Gaussians through our neural velocity, allowing them to efficiently represent dynamics as a series of piecewise linear movements. Second, we modify our temporal opacity based on a generalized Gaussian function. This modification captures sudden changes in dynamic scenes more effectively, whereas the temporal opacity of previous methods [10, 14] lack this capability. These formulations enhance the capability of Gaussians to capture dynamics in local regions, reducing the number of anchors required for high quality.

Our 4D anchor-based reconstruction pipeline effectively addresses redundancy present in dynamic scenes through compression. As shown in Figure 1, our method achieves state-of-the-art visual quality and rendering speed, while reducing storage demands by 93.8% compared to 4DGS [36].

## 2. Related work

This section presents a comprehensive review of scene representations, focusing on three aspects. First, we examine approaches for efficient 3D Gaussians (Sec. 2.1). Then, we investigate methods for both deforming 3D Gaussians and utilizing 4D Gaussians in dynamic scene reconstruction

(Sec. 2.2). Lastly, we review various approaches employing neural features for scene reconstruction (Sec. 2.3).

### 2.1. Efficient 3D Gaussians

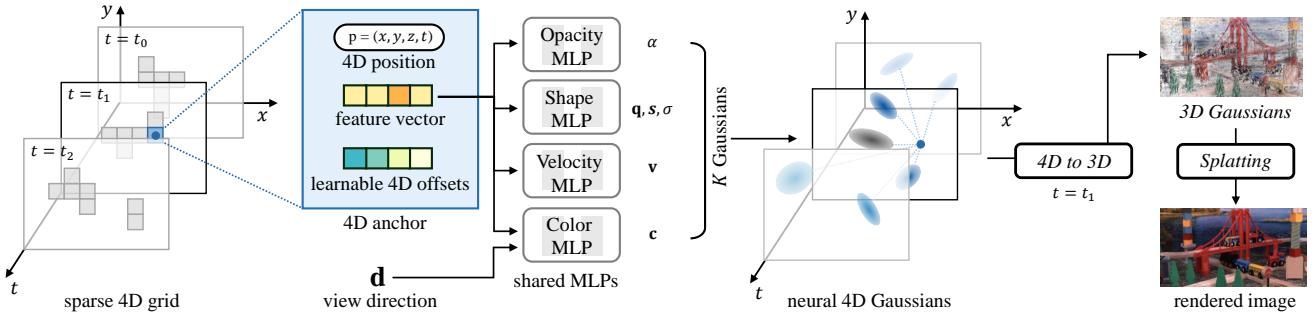
In recent years, 3D Gaussian Splatting (3DGS) has attracted significant attention for achieving real-time rendering by representing scenes with 3D Gaussian primitives and introducing a tile-based rasterizer. For photorealistic rendering quality, 3DGS requires a significant number of 3D Gaussians, which increases storage costs. To address this problem, [6, 7, 29] remove unnecessary Gaussians that do not harm rendering quality, [16, 19, 20] replace Gaussians or their parameters with efficient representations, and [4, 18, 19, 33] compress Gaussian parameters using existing compression techniques such as entropy coding and image compression. In this paper, we address eliminating excess Gaussians and improving efficiency in dynamic scenes with 4D Gaussians, which have been relatively unexplored compared to 3D Gaussians.

### 2.2. Dynamic 3D Gaussians

Two main approaches are proposed to extend 3DGS [8] into dynamic scene reconstruction. The first approach involves deforming 3D Gaussians along with temporal changes [2, 15, 26, 31, 34]. These deformable 3D Gaussians offer the advantage of compact storage requirements but exhibit relatively slow rendering speeds and low visual quality. In contrast, the other approach directly employs 4D Gaussians in the spatio-temporal domain [14, 35]. These approaches demonstrate superior visual quality and faster rendering speeds, but suffer from higher storage requirements. To address this, our method efficiently models temporal opacity and predicts Gaussian properties through neural features.

### 2.3. Feature-based neural rendering

Recently, a growing trend in scene reconstruction has been to integrate neural features as additional inputs to enhance model performance. For instance, there have been attempts to extract features from source views and utilize them for novel view synthesis, enabling few-view reconstruction [3, 37] or view interpolation through transformers [22, 27, 30]. In 3DGS studies, [5] uses a hash grid instead of per-Gaussian SH coefficients, [14] renders features and decodes them into RGB colors via shallow MLP. [32] generates Gaussian attributes from a multi-level tri-plane. [16] predicts the attributes of local 3D Gaussians from anchor features. Our work focuses on efficient dynamic representations, which are relatively underexplored. We introduce a 4D anchor-based framework that includes dynamic neural velocity and temporal opacity derived from a generalized Gaussian distribution, considering real-world dynamics.



**Figure 2. Overview of 4D anchor-based method.** We begin with a sparse 4D grid of anchors, each defined by a unique 4D spatiotemporal position  $p$  and learnable offsets. Shared MLPs utilize these anchors to generate neural 4D Gaussians, capturing dynamic appearance changes with view direction  $d$ . These 4D Gaussians are then projected to 3D Gaussians at specific timesteps for rendering, using a 3D Gaussian splatting pipeline to produce the final rendered image. We omit the  $z$ -axis for simplicity.

### 3. Method

We reconstruct dynamic 3D scenes with 4D Gaussians. Especially, our method builds 4D anchors which specify nearby 4D Gaussians (Sec. 3.1). We encourage anchors to be closely placed at under-reconstructed dynamic regions (Sec. 3.2). Furthermore, we design each 4D Gaussian to move along a line segment, and define the temporal opacity function which fits piecewise persistent period (Sec. 3.3).

#### 3.1. Overview

The overview of our method is illustrated in Figure 2. Our method begins with a set of sparse 4D anchor points, each of which has a unique, grid-aligned spatiotemporal 4D position  $p \in \mathbb{R}^4$  with a feature vector  $f \in \mathbb{R}^C$ . We leverage shared MLPs to produce  $K$  neural 4D Gaussians from these anchor features. Corresponding 3D Gaussians are computed from these 4D Gaussians to render a frame at timestep  $t$ . We describe details of our method in the following paragraphs.

**Initializing 4D anchors.** Similar with 3D Scaffold-GS [17], we utilize static point clouds to initialize 4D anchor points. We first obtain the static point cloud from multi-view frames at a certain timestep  $t$  using Structure-from-Motion (SfM) [25]. The 4D positions of anchors are initialized from a set of voxels  $V \subset \mathbb{R}^{N \times 3}$  obtained from this point cloud:

$$p_v = (x_v, y_v, z_v, t), \quad \forall v \in V, \quad (1)$$

where  $(x_v, y_v, z_v)$  is the center coordinates of the voxel  $v$ . Each anchor point also accompanies two learnable parameters: a feature vector  $f$ , and 4D offsets  $o \in \mathbb{R}^{K \times 3}$  for determining positions of  $K$  neural 4D Gaussians.

Beginning with 4D anchors initialized with the static point cloud, our method gradually increases anchors to dynamic regions. This is accomplished by our anchor growing strategy, which will be described in Sec. 3.2.

**Neural 4D Gaussians.** We leverage shared MLPs and

learnable parameters of the 4D anchors to generate  $K$  neural 4D Gaussians from each anchor. Specifically, shared MLPs take the anchor feature  $f$  and yield properties of  $K$  neural Gaussians, which include time-invariant opacity  $\alpha \in \mathbb{R}$ , quaternion  $q \in \mathbb{R}^4$  and scaling  $s \in \mathbb{R}^3$  for the covariance matrix, view-dependent color  $c \in \mathbb{R}^3$ . Our shared MLPs also produce temporal scale  $\sigma \in \mathbb{R}$  to compute our temporal opacity, and the neural velocity  $v \in \mathbb{R}^3$ . Note that the color MLP additionally takes the view direction  $d \in \mathbb{R}^3$  as inputs for view-dependent modeling.

**Rendering neural Gaussians.** To render the neural 4D Gaussians, we compute 3D Gaussian parameters at time  $t$  from our time-varying properties. For the  $k$ -th Gaussian of the anchor  $p$ , the opacity  $\alpha_k(t)$  and the center  $\mu_k(t)$  of the corresponding 3D Gaussian at time  $t$  are derived as

$$\mu_k(t) = \mu_{k,1:3} + h(t, \mu_{k,4}, v), \quad (2)$$

$$\alpha_k(t) = \alpha_k \cdot g(t, \mu_{k,4}, \sigma_k), \quad (3)$$

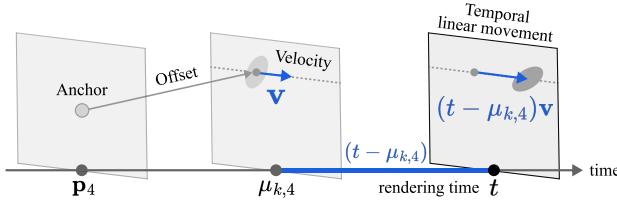
where  $\mu_k = p + o_k$  is the coordinates of the 4D Gaussian.  $h(\cdot)$ ,  $g(\cdot)$  model time-varying values of the positions and the opacity with the neural velocity and temporal opacity function, which will be further described in Sec. 3.3.

After deriving 3D Gaussians at time  $t$ , we utilize the existing efficient 3D Gaussian splatting pipeline [8] to render the frames. Same as 3D Scaffold-GS [17], Gaussians within the view frustum and having opacity higher than the threshold are passed to the renderer.

#### 3.2. Temporal coverage-aware anchor growing

Growing new anchors to under-reconstructed dynamic regions is a critical factor for high-quality results. The direct application of previous anchor growing to the dynamic reconstruction fails to identify such regions, as it neglects the temporal coverage of the Gaussians.

To this end, we propose a temporal coverage-aware anchor growing operation. The core of our anchor growing is



**Figure 3. Time-variant Gaussian position calculation.** The 3D Gaussian position at rendering time is computed as the sum of the anchor position, the 4D Gaussian offset, and the temporal linear movement. The temporal linear movement is the product of the neural velocity and the time difference, allowing the Gaussian to traverse 3D space over time.

the computation of the accumulated gradients of each 4D Gaussian  $\nabla_g$ . We formulate  $\nabla_g$  as a weighted sum of the gradient over  $N$  iterations:

$$\nabla_g = \frac{\sum^N w(t, \sigma) \|\nabla_{2D}\|}{\sum^N w(t, \sigma)}, \quad (4)$$

where  $\nabla_{2D}$  is the 2D position gradient of the Gaussian. We define the weight term  $w$  as a function of  $\alpha(t)$  and  $\sigma$ , with  $\gamma$  as a hyperparameter:

$$w(t, \sigma) = \alpha(t)(1/\sigma)^\gamma. \quad (5)$$

We collect the accumulated gradients of all Gaussians and then voxelize them. New anchors are placed at the centers of the voxels having gradients higher than a threshold.

In contrast, the previous anchor growing strategy [17] designed for static scenes gathers the gradients as a mean over  $N$  iterations:

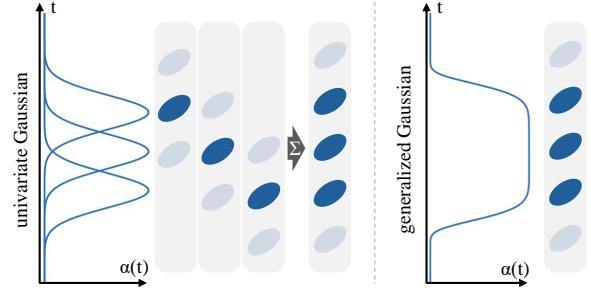
$$\nabla_g = \frac{\sum^N \|\nabla_{2D}\|}{N}. \quad (6)$$

With this strategy, dynamic regions appearing in a short period will have lower  $\nabla_g$ , as these regions will be penalized by the denominator  $N$  regardless of their actual errors.

The first difference of our approach compared to the previous anchor growing strategy [17] is that the gradients are accumulated only when the Gaussian is activated. Our approach accurately gathers the gradients of 4D Gaussians placed at the regions where dynamic scene elements only appear in a short period. The second difference is that the temporal coverage  $\sigma$  directly influences to the anchor growing operation. This makes regions having a short temporal coverage easier to grow. By modifying the accumulated gradients as a weighted sum, our approach effectively places new anchors to under-reconstructed dynamic regions.

### 3.3. Modeling 4D Gaussians for compact anchors

To represent dynamic scenes with fewer anchors, each neural Gaussian should be able to model the scene element in a



**Figure 4. Illustration of the modified temporal opacity.** To reconstruct the sudden and continuous appearance of an object, the previous univariate Gaussian model requires the sum of multiple Gaussians, while the generalized Gaussian achieves the same expressiveness with only a single Gaussian.

sufficient time range. This necessitates neural 4D Gaussians to accurately model the real-world dynamics.

We introduce two key formulations of 4D Gaussians that improve the capability of Gaussians of modeling real-world dynamics. We model local movements through the neural velocity, and we modify the temporal opacity based on a generalized Gaussian distribution.

**Neural velocity.** In computer graphics, one common way to model the motion of an object is through a set of piecewise linear segments. Based on this concept, we propose the neural velocity to model the motions of scene elements with a set of 4D Gaussians having linear movements (see Figure 3). As described in Equation (2), the time-varying 3D position of the 3D Gaussian  $\mu_k(t) \in \mathbb{R}^3$  is determined by the time-varying function  $h(\cdot)$ . We formulate the  $h(\cdot)$  as the linear movement based on the neural velocity  $v$ :

$$h(t, \mu_{k,4}, v) = (t - \mu_{k,4})v. \quad (7)$$

Note that our neural velocity simplifies the temporal slicing of 4D positions in 4DGS [10], without requiring excessive parameters for 4D covariances.

Despite its simplicity, our neural velocity enables each 4D Gaussian to cover dynamic movements of the elements in a sufficient time range. Such capability reduces the number of neural 4D Gaussians and anchors as a result.

**Modified temporal opacity.** Previous 4D Gaussian methods [10, 14] adopt the univariate Gaussian function as temporal opacity to make Gaussians appear within a specific time range. However, this univariate Gaussian lacks the ability to model sudden changes in real-world scene elements, resulting in excessive Gaussians to model a single scene element. This issue is also discussed in the concurrent work [11], which proposes a simplified Gaussian mixture model as a solution.

Therefore, we propose to modify the temporal opacity function to remove such redundant Gaussians (see Figure 4). We reformulate the temporal opacity function based

Table 1. **Quantitative results on N3DV dataset.** <sup>1</sup> only includes the *flame\_salmon* for storage and FPS [2]. <sup>2</sup> divides the scene into 50 frames and trains each using a separate model. <sup>3</sup> uses COLMAP [25] for all 300 frames.

Model	PSNR↑	SSIM↑	LPIPS↓	Storage↓	FPS↑
K-Planes [24] <sup>1</sup>	31.22	0.947	0.090	309 MB	0.13
MixVoxels-L [28] <sup>1</sup>	30.81	0.933	0.095	512 MB	0.93
HyperReel [1] <sup>1 2</sup>	31.10	0.927	0.099	1362 MB	1.04
4DGaussian [31]	30.71	0.935	0.056	<b>59 MB</b>	51.9
E-D3DGS [2]	31.42	0.945	<b>0.037</b>	137 MB	19.9
STG [14] <sup>2 3</sup>	31.96	<b>0.948</b>	0.046	173 MB	125.9
4DGS [35]	<b>32.14</b>	0.947	0.047	6194 MB	61.4
Ours	<u>32.03</u>	<u>0.947</u>	<u>0.041</u>	<u>134 MB</u>	<b>129.9</b>

on a generalized Gaussian distribution. This offers steeper derivatives at the beginning and the end, making it better suited to fit piecewise persistent periods. Our time-varying opacity  $g(\cdot)$  in Equation (3) is formulated as

$$g(t, \mu_{k,4}, \sigma_k) = e^{-(|t - \mu_{k,4}|/\sigma_k)^\beta}. \quad (8)$$

This formulation greatly reduces the number of 4D Gaussians and anchors to represent dynamic scenes while preserving the visual quality. In practice, we set the inverse sigma as a learnable parameter, as it leads to more stable training. We also model  $\beta = 2\beta'$  and  $\beta'$  as a hyperparameter, which removes the  $|\cdot|$  in the above equation.

### 3.4. Training and implementation details

**Training objective.** Both learnable anchor parameters and MLP weights are jointly optimized through the rendering loss. We employ  $\mathcal{L}_1$  with SSIM loss  $\mathcal{L}_{\text{SSIM}}$  and volume regularization  $\mathcal{L}_{\text{vol}}$  following the 3D Scaffold-GS [16]. The full training objective with weighting coefficients  $\lambda_{\text{SSIM}} = 0.2$  and  $\lambda_{\text{vol}} = 0.01$  is

$$\mathcal{L} = (1 - \lambda_{\text{SSIM}})\mathcal{L}_1 + \lambda_{\text{SSIM}}\mathcal{L}_{\text{SSIM}} + \lambda_{\text{vol}}\mathcal{L}_{\text{vol}}. \quad (9)$$

**Implementation details.** Similar to 4DGS, our model is trained for 120K iterations with a single batch, which takes approximately 4 hours on a single NVIDIA A6000 GPU. We set  $\beta = 2$  and  $\gamma = 1$  for our modified temporal opacity model and temporal coverage-aware densification. For 4D voxel grid size, We use 0.001 as spatial grid size and use frame interval time as temporal grid size for all scenes. After training, we prune the unnecessary anchor points of which all Gaussians have opacity near zero. To improve rendering speed, we cache the time-invariant and view-invariant outputs of MLPs. We provide more implementation details in the supplements.

## 4. Experiments

In this section, we first evaluate our method through the comparisons with several state-of-the-art baselines for dy-

Table 2. **Quantitative results on Technicolor dataset.** <sup>1</sup> refers to the values reported in [14]. <sup>2</sup> only includes the *Painter* for storage and FPS [2]. <sup>3</sup> uses COLMAP [25] for all 50 frames.

Model	PSNR↑	SSIM↑	LPIPS↓	Storage↓	FPS↑
DyNeRF [12] <sup>1</sup>	31.80	-	0.140	<b>30 MB</b>	0.02
HyperReel [1] <sup>2</sup>	32.32	0.899	0.118	289 MB	0.91
4DGaussian [31]	29.62	0.844	0.176	<u>51 MB</u>	34.8
E-D3DGS [2]	33.24	0.907	0.100	<u>77 MB</u>	36.5
STG [14] <sup>2 3</sup>	<u>33.40</u>	<u>0.918</u>	<u>0.082</u>	55 MB	112.7
4DGS [35]	33.35	0.910	0.095	9106 MB	43.3
Ours	<b>33.97</b>	<b>0.923</b>	<b>0.074</b>	98 MB	<b>124.5</b>

namic scene reconstruction. Then we conduct ablations and analysis to explore the effectiveness of our main features. Our code will be made publicly available.

**Datasets.** We evaluate our method on two representative real-world datasets: Neural 3D Video dataset (N3DV) [12] and Technicolor dataset [23].

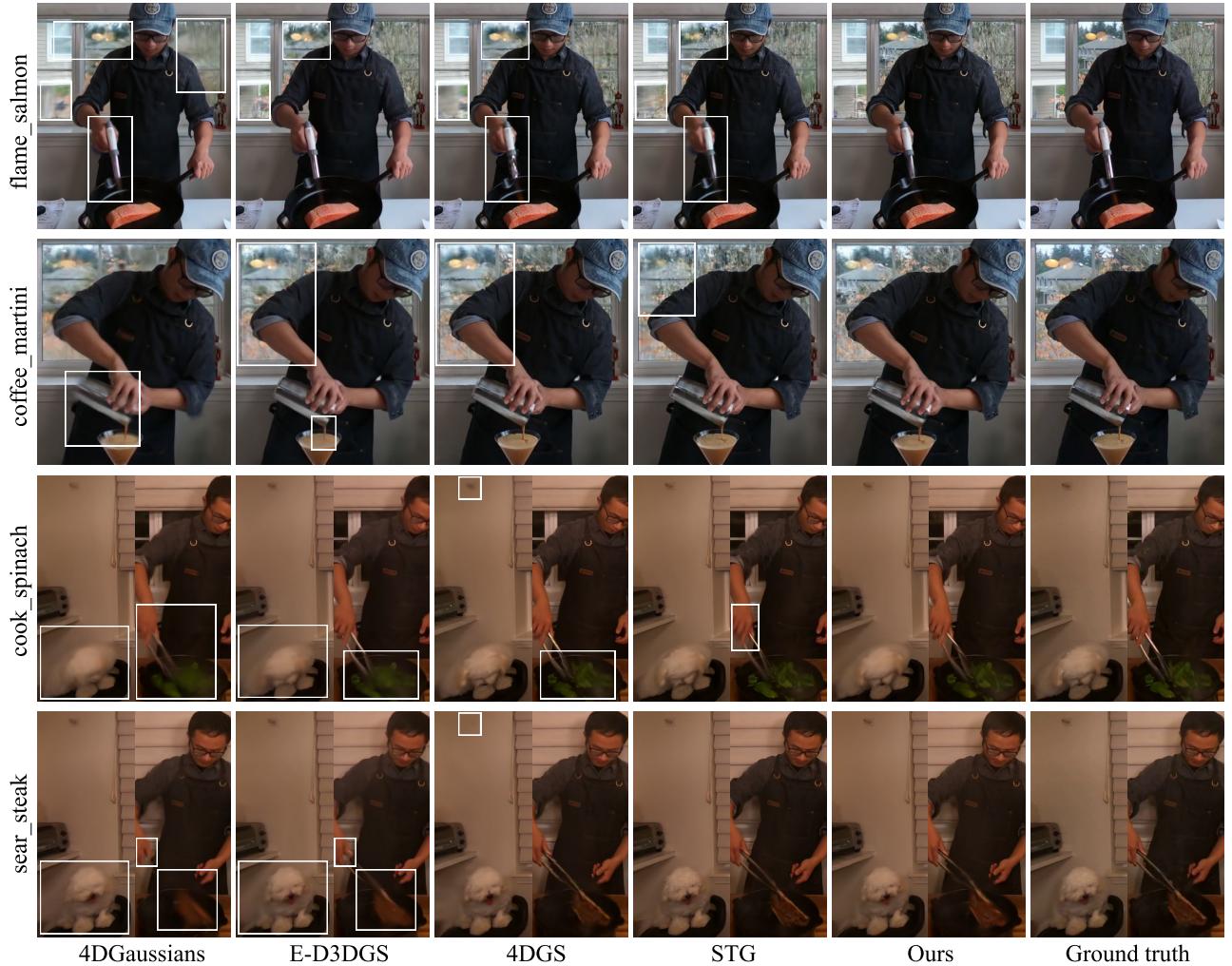
- **Neural 3D Video dataset (N3DV)** comprises 17 to 21 synchronized videos of six scenes, with each video containing 300 frames. Following previous works, we downsample the resolution to  $1352 \times 1014$  and use the first camera as the test view. We exclude *cam13* for the *coffee\_martini* scene due to synchronization issues.
- **Technicolor Dataset** contains 16 synchronized videos across five scenes, with each video comprising 50 frames. We retain the original resolution of  $2048 \times 1088$  and use *cam10* as the test view.

**Baselines.** We choose the following state-of-the-art competitors: 4DGaussian [31] and E-D3DGS [2] representing deformation-based methods, and 4DGS [35] and STG [14] representing 4D Gaussians. We reproduce them using their official code to report their performance. We also include representatives of NeRF-based methods: K-Planes [24], MixVoxels [28], and HyperReel [1]. Results of these methods are brought from [2].

**Metrics.** To assess the reconstruction quality, we measure peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), and LPIPS [38] of the rendered images. We employ AlexNet [9] to compute LPIPS following the previous papers. To evaluate storage efficiency, we calculate the total size of output files, including MLP weights and parameters of Gaussians or anchors. We measure frames-per-second (FPS) by recording the wall-clock rendering time on the same machine, with a single NVIDIA A6000 GPU.

### 4.1. Reconstruction quality

**N3DV.** Table 1 presents the quantitative results on the N3DV dataset. Our method outperforms all NeRF baselines in both visual quality and rendering speed. It also achieves competitive results in visual quality compared to state-of-



**Figure 5. Qualitative comparisons on the Neural 3D Video dataset.** This figure provides qualitative comparisons on the Neural 3D Video dataset. White boxes indicate under-reconstructed areas. Our method achieves high fidelity in both dynamic and static regions.

the-art Gaussian Splatting-based methods, and offers the fastest rendering speed across all methods. Remarkably, our method significantly reduces storage costs by  $97.8\times$  compared to 4DGS [35], while achieving comparable or better performance in visual quality and rendering speed. It is worth noting that STG requires COLMAP for all frames, and train 6 models to obtain its results. We provide further comparisons with STG in the supplements.

To further assess the performance, we deliver the qualitative comparisons with Gaussian Splatting baselines in Figure 5. Our method successfully reconstructs both static and dynamic regions. In contrast, other baselines often struggle to reconstruct dynamic regions (see the boxed regions). Deformation-based methods (4DGaussians, E-D3DGS) tend to produce blurry results, and the results of 4D Gaussian-based methods (4DGS, STG) often contain artifacts in dynamic regions. By effectively modeling

dynamic regions with fewer anchors, our method delivers high-quality results in an efficient manner. Please refer to the supplements for more visual comparisons, including uncropped results and videos.

**Technicolor.** We deliver the quantitative comparisons on the Technicolor dataset in Table 2. Our method surpasses all baselines in visual quality and rendering speed. We also achieve  $85.1\times$  storage reduction and  $2.2\times$  improvement in FPS, compared to 4DGS. Although deformation-based methods offer compact storage usage, these methods require expensive computations for Gaussian deformations. This leads to the low FPS of these methods.

Figure 6 presents the qualitative comparisons with Gaussian-based methods on the Technicolor dataset. Our method produces sharp and high-fidelity rendering results in both static and dynamic regions. On the other hand,

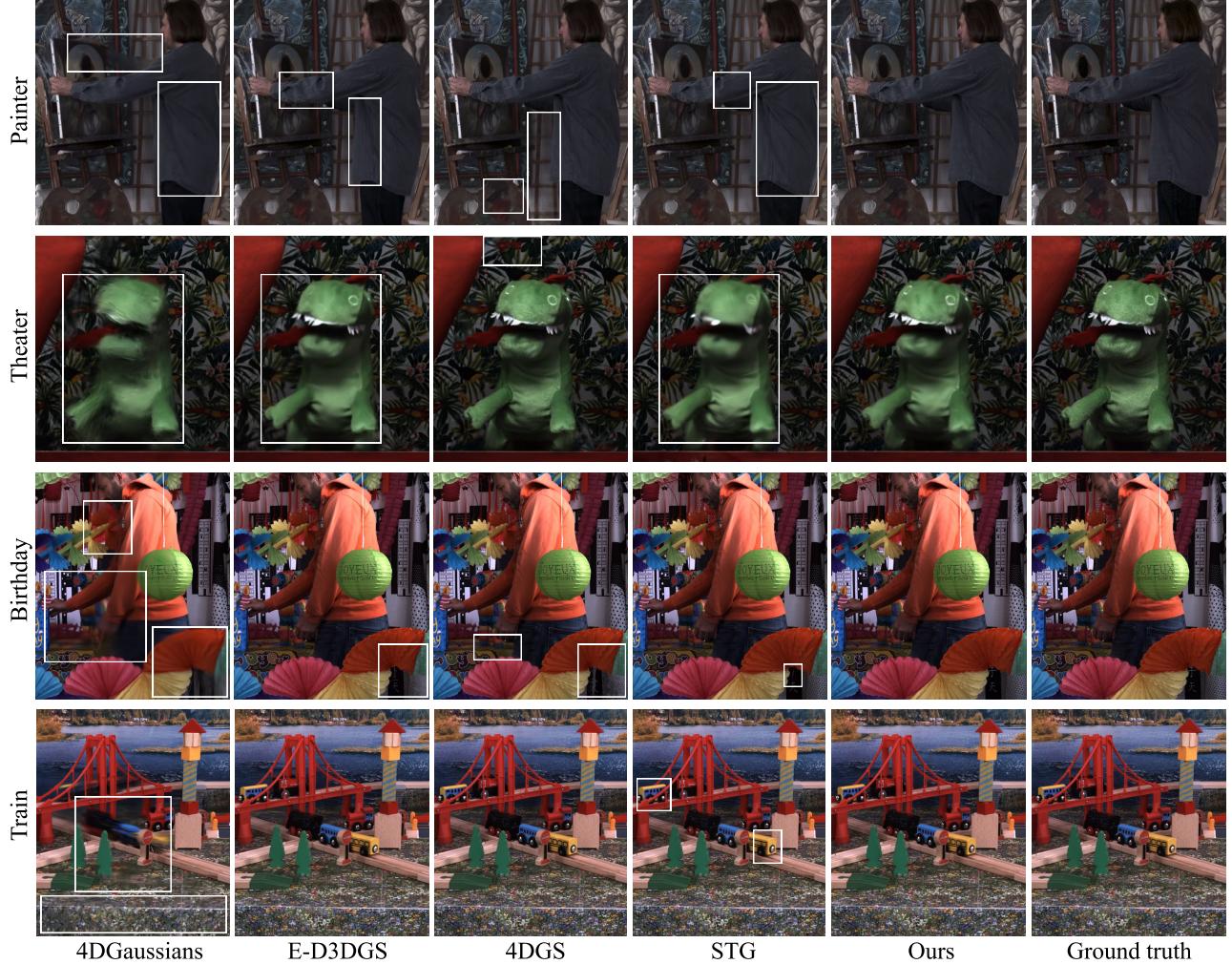


Figure 6. **Qualitative comparisons on the Technicolor dataset.** This figure provides qualitative comparisons on the Technicolor dataset. White boxes indicate under-reconstructed areas. Our method achieves high fidelity in both dynamic and static regions.

4DGaussians, E-D3DGS and STG show blurry results in dynamic regions, such as the boxed part in the *Theater* scene. E-D3DGS and STG often fail to reconstruct several details, *e.g.* the eye in the *Theater* scene.

## 4.2. Analysis and ablation study

To provide deeper insights of our main features, we conduct the ablations and analysis on the N3DV dataset. Please refer to the supplements for more detailed analysis and ablations on each component.

**Effects of each feature on quality and storage.** We verify the effectiveness of our proposals through the ablations. We begin with the baseline model (Baseline) that does not employ none of our main components. Then we gradually add each component to the baseline and report its results.

As shown in Table 3, our model with all components achieves the best results in visual quality with reasonable

storage. Adding the temporal coverage-aware anchor growing operation improves visual quality but increases the anchors. The neural velocity reduces the number of anchors but brings degradation in visual quality. Finally, modifying the temporal opacity achieves the highest visual quality with fewer anchors. These results support that our anchor growing strategy successfully allocates new anchors to under-reconstructed dynamic regions, improving visual quality. Appropriate modeling of neural Gaussians reduces the number of anchors without sacrificing performance.

**Understanding anchor growing.** To provide an in-depth understanding of the proposed anchor growing strategy, we present visual comparisons with previous anchor growing operations. We visualize the accumulated gradients  $\nabla_g$  of both methods during training. We also compare resulting anchors and rendered images after training.

Table 3. **Ablation results on the main components.** The abbreviations are as follows: “TA” refers to the temporal coverage-aware anchor growing, “NV” to the neural velocity and “MT” to the modified temporal opacity.

TA	NV	MT	PSNR↑	LPIPS↓	#Anchor↓	Storage↓
✓			31.20	0.053	207K	83 MB
✓	✓		31.74	0.043	888K	301 MB
✓	✓		31.11	0.042	576K	195 MB
✓		✓	31.82	0.043	793k	268 MB
✓	✓	✓	32.03	0.041	440K	149 MB

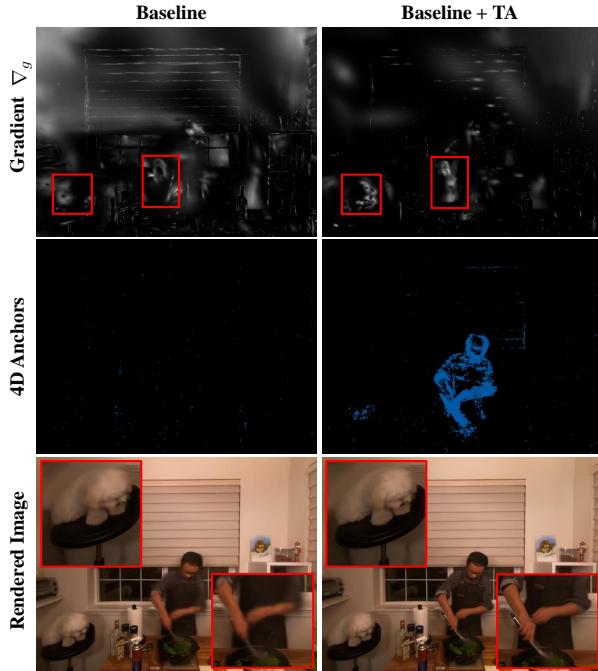


Figure 7. **Anchor growing analysis.** Top-to-bottom: accumulated gradients at 5000th training iteration, final 4D anchors at 106 frame, and the rendering results. Our anchor growing effectively accumulates gradients in under-reconstructed dynamic regions.

Figure 7 presents visual comparisons with the previous anchor growing operation. The previous method fails to accurately collect gradients in dynamic regions due to their short appearing periods. This leads to a lack of anchors in under-reconstructed regions, resulting in degraded visual quality. In contrast, our method successfully accumulates higher gradients in these dynamic regions (see the red-boxed parts in Figure 7). Consequently, the anchors generated by our method more accurately represent dynamic regions at each timestep. These results demonstrate that our anchor-growing strategy effectively captures under-reconstructed dynamic regions, playing a critical role in achieving high-quality reconstruction.

**Understanding temporal opacity.** We explore the effects of the temporal opacity by observing actual distributions.

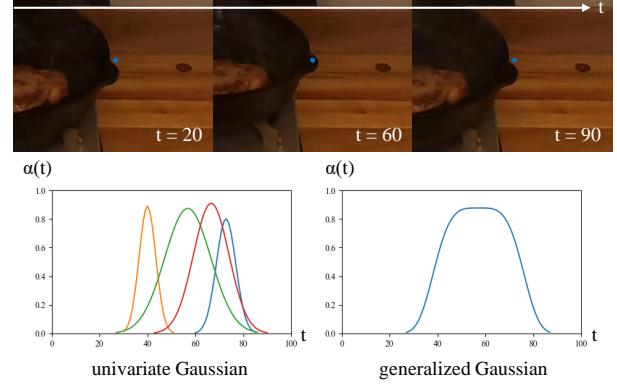


Figure 8. **Temporal opacity analysis.** We visualize Gaussians that contributing to the rendering of a sample pixel in the *sear\_steak* scene from the N3DV dataset. For clarity, we filter out Gaussians distant from the frying pan based on depth and image-plane probability. Our modified temporal opacity better aligns with real-world dynamics, allowing a single Gaussian to represent the element.

Specifically, we choose an example pixel in *sear\_steak*, which contains an object that appears in a certain period (the frying pan). We visualize Gaussians that primarily contribute to render this pixel. We compare our temporal opacity with the previous one based on a univariate Gaussian function.

As shown in Figure 8, the resulting Gaussian with our modified temporal opacity better fits the actual distributions of the object. This enables our method to represent a scene element with a single Gaussian. On the other hand, previous univariate Gaussian is not well-aligned to real-world dynamics, resulting in multiple Gaussians representing the same element. As a result, our temporal opacity reduces the number of anchors while retaining visual quality, which leads to compact storage usage.

## 5. Conclusion

We proposed the 4D anchor-based method for dynamic scene reconstruction. Our anchor growing and neural 4D Gaussians modeling capture dynamic regions with an efficient number of anchors. Experimental results support the validity of our method, achieving compelling visual quality and FPS while significantly reducing storage consumption.

**Limitations and future work.** Since our method is currently designed for multi-view video datasets, applying it on monocular videos can introduce additional challenges. Same as other methods, our method still suffer from reconstructing elements that appear very shortly (1 or 2 frames). Resolving this challenge can be an interesting future work.

## References

- [1] 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 (CVPR)*, pages 16610–16620, 2023. [1](#), [5](#)
- [2] Jeongmin Bae, Seoha Kim, Youngsik Yun, Hahyun Lee, Gun Bang, and Youngjung Uh. Per-gaussian embedding-based deformation for deformable 3d gaussian splatting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024. [1](#), [2](#), [5](#)
- [3] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14124–14133, 2021. [2](#)
- [4] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac: Hash-grid assisted context for 3d gaussian splatting compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 422–438. Springer, 2025. [2](#)
- [5] Tianchen Deng, Yaohui Chen, Leyan Zhang, Jianfei Yang, Shenghai Yuan, Jiuming Liu, Danwei Wang, Hesheng Wang, and Weidong Chen. Compact 3d gaussian splatting for dense visual slam. *arXiv preprint arXiv:2403.11247*, 2024. [2](#)
- [6] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023. [2](#)
- [7] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eangles: Efficient accelerated 3d gaussians with lightweight encodings. *arXiv preprint arXiv:2312.04564*, 2023. [2](#)
- [8] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4), 2023. [1](#), [2](#), [3](#)
- [9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 25, 2012. [5](#)
- [10] Junoh Lee, ChangYeon Won, Hyunjun Jung, Inhwon Bae, and Hae-Gon Jeon. Fully explicit dynamic guassian splatting. In *Proceedings of the Neural Information Processing Systems (NeurIPS)*, 2024. [2](#), [4](#)
- [11] Junoh Lee, ChangYeon Won, Hyunjun Jung, Inhwon Bae, and Hae-Gon Jeon. Fully explicit dynamic guassian splatting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. [4](#)
- [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 (CVPR)*, 2022. [1](#), [5](#)
- [13] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. [1](#)
- [14] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. Spacetime gaussian feature splatting for real-time dynamic view synthesis. *arXiv preprint arXiv:2312.16812*, 2023. [1](#), [2](#), [4](#), [5](#)
- [15] Jiahao Lu, Jiacheng Deng, Ruijie Zhu, Yanzhe Liang, Wenfei Yang, Tianzhu Zhang, and Xu Zhou. Dn-4dgs: Denoised deformable network with temporal-spatial aggregation for dynamic scene rendering. *arXiv preprint arXiv:2410.13607*, 2024. [2](#)
- [16] 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 (CVPR)*, pages 20654–20664, 2024. [2](#), [5](#), [3](#)
- [17] 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 (CVPR)*, pages 20654–20664, 2024. [2](#), [3](#), [4](#)
- [18] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. *arXiv preprint arXiv:2312.13299*, 2023. [2](#)
- [19] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10349–10358, 2024. [2](#)
- [20] 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. [2](#)
- [21] 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), 2021. [1](#)
- [22] Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction, 2021. [2](#)
- [23] Neus Sabater, Guillaume Boisson, Benoit Vandame, Paul Kerbiriou, Frederic Babon, Matthieu Hog, Remy Gendrot, Tristan Langlois, Olivier Bureller, Arno Schubert, et al. Dataset and pipeline for multi-view light-field video. In *Proceedings of the IEEE conference on computer vision and pattern recognition Workshops*, pages 30–40, 2017. [5](#)
- [24] Sara Fridovich-Keil and 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 (CVPR)*, 2023. [1](#), [5](#)
- [25] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the*

*IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 3, 5

- [26] Richard Shaw, Michal Nazarczuk, Jifei Song, Arthur Moreau, Sibi Catley-Chandar, Helisa Dhamo, and Eduardo Pérez-Pellitero. Swings: sliding windows for dynamic 3d gaussian splatting. Proceedings of the European Conference on Computer Vision (ECCV), 2024. 2
- [27] Mukund Varma T, Peihao Wang, Xuxi Chen, Tianlong Chen, Subhashini Venugopalan, and Zhangyang Wang. Is attention all that neRF needs? In *The Eleventh International Conference on Learning Representations*, 2023. 2
- [28] Feng Wang, Sinan Tan, Xinghang Li, Zeyue Tian, and Huaping Liu. Mixed neural voxels for fast multi-view video synthesis. *arXiv preprint arXiv:2212.00190*, 2022. 1, 5
- [29] Henan Wang, Hanxin Zhu, Tianyu He, Runsen Feng, Jianjun Deng, Jiang Bian, and Zhibo Chen. End-to-end rate-distortion optimized 3d gaussian representation. *arXiv preprint arXiv:2406.01597*, 2024. 2
- [30] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2
- [31] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Wang Xinggang. 4d gaussian splatting for real-time dynamic scene rendering. *arXiv preprint arXiv:2310.08528*, 2023. 1, 2, 5
- [32] Minye Wu and Tinne Tuytelaars. Implicit gaussian splatting with efficient multi-level tri-plane representation. *arXiv preprint arXiv:2408.10041*, 2024. 2
- [33] Shuzhao Xie, Weixiang Zhang, Chen Tang, Yunpeng Bai, Rongwei Lu, Shijia Ge, and Zhi Wang. Mesongs: Post-training compression of 3d gaussians via efficient attribute transformation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 434–452. Springer, 2025. 2
- [34] 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. 1, 2
- [35] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *International Conference on Learning Representations (ICLR)*, 2024. 1, 2, 5, 6
- [36] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *International Conference on Learning Representations (ICLR)*, 2024. 2
- [37] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2
- [38] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–595, 2018. 5

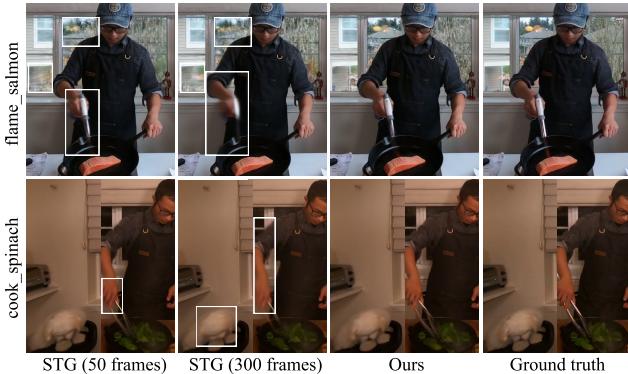
# 4D Scaffold Gaussian Splatting for Efficient Dynamic Scene Reconstruction

## Supplementary Material

### A. Additional comparison with STG

As mentioned in the main paper, it is difficult to fairly compare with Spacetime Gaussians (STG) [14] due to the different experimental setups. Unlike our method, STG utilizes per-frame COLMAP to obtain the initial points and divide long video sequences into 50-frame chunks, training separate models for each chunk. This results in six separate models to report its results on N3DV.

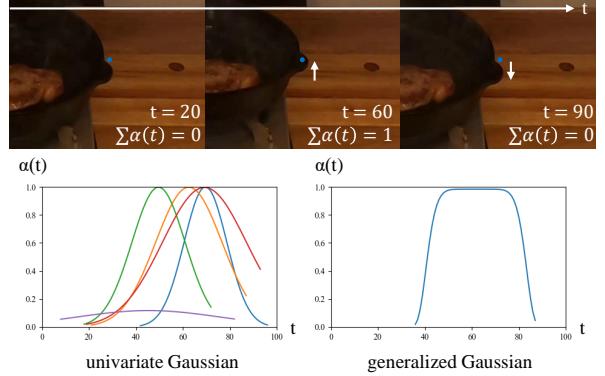
We additionally deliver the results of STG on N3DV, under the same experimental setup in Figure S1 and Table S1. While STG is effective on shorter video sequences, it often fails to accurately reconstruct dynamic regions on longer sequences. On the other hand, our method effectively represents dynamic regions on longer sequences. As a result, our method achieves better visual quality than STG under the same experimental setups.



**Figure S1. Comparison between 300-frame STG and Ours.** The single STG model trained on 300 frames of the N3DV dataset shows lower quality in the dynamic regions compared to Ours. The white boxes highlight under-reconstruction areas.

**Table S1. Comparison with STG on the N3DV dataset.** Our model outperforms the average of six STGs trained on 50 frames and a single STG trained on 300 frames. \* Note that STG uses a batch size of 2 while our model uses a batch size of 1.

	# models	# iterations	PSNR↑	LPIPS↓
STG	6 (50 frames)	6×30K*	31.96	0.046
	1 (300 frames)	60K*	31.25	0.053
Ours	1 (300 frames)	120K	<b>32.03</b>	<b>0.041</b>



**Figure S2. Impact of modified temporal opacity with 4DGs.** We apply our modified temporal opacity to 4DGs and analyze its behavior on the *sear\_steak* scene from N3DV. The total temporal opacity distribution for the frying pan (blue point) is low at  $t = 20$  and  $t = 90$  while peaking at  $t = 60$ , with rapid transitions in between. Each plot represents the temporal opacity of an individual Gaussian. Unlike conventional methods relying on Gaussian mixtures, our generalized Gaussian effectively captures this behavior using a single Gaussian component.

### B. Modified temporal opacity with 4DGs.

We further validate the effectiveness of our modified temporal opacity by applying it to 4DGs [35]. Consistent with the main paper, we visualize temporal opacity values at the same points within *sear\_steak* scene of the N3DV dataset.  $\beta = 8$  is used for the generalized Gaussian equation.

As shown in Figure S2, the previous univariate Gaussian function struggles to represent elements that suddenly appear and disappear. This limitation results in five Gaussians to model the same point. On the other hand, our modified temporal opacity effectively represents these dynamic changes, enabling the representation with only one Gaussian. Consequently, 4DGs with the modified temporal opacity achieves a storage reduction from 4.74 GB to 3.82 GB.

### C. Additional ablation study

**Modeling 4D Gaussians without anchors.** To analyze the effectiveness of our anchor-based reconstruction pipeline, we evaluate the performance of our model without the scaffold structure. Specifically, we replace all MLP outputs with explicit learnable Gaussian parameters. As shown in Table S2, removing the scaffold structure leads to a degradation in visual quality. The structured anchor representations offer more stable training and anchor growing, which significantly improves the performance of our method.

**Polynomial Gaussian trajectory.** Our neural velocity represents the local dynamics of Gaussians as linear movements. Despite its simplicity, these linear movements sufficiently represent dynamics as 3D scene flows. We explore the effectiveness of the neural velocity by replacing it with a polynomial trajectory proposed in the previous method [14]. We use  $n_p = 3, n_q = 1$  as they use for their full model. The velocity MLP outputs coefficients of the polynomial equation instead of the neural 3D velocity. As shown in Table S2, this approach results in decreased performance and increased storage. Since the polynomial trajectory requires 12 parameters per Gaussian, a total of 120 parameters ( $K = 10$ ) as MLP outputs, it often complicates the training of our lightweight MLP and leads to degradation.

Table S2. **Additional model ablations.** Quantitative results on different model designs on the N3DV dataset. Our full method utilizes scaffold structure and neural velocity.

Scaffold	Movement	PSNR↑	LPIPS↓	Storage↓
	velocity	30.11	0.060	393 MB
✓	polynomial	31.68	0.046	323 MB
✓	velocity	32.03	0.041	134 MB

## D. Effects of hyperparameters

We investigate the impact of hyperparameters on the performance and their controllability. *cook\_spinach* and *flame\_salmon* scenes from N3DV are selected for analysis.

**Temporal coverage-aware anchor growing.** The parameter  $\gamma$  in Equation (5) controls the influence of temporal scale on densification. As  $\gamma$  increases, densification becomes more concentrated in dynamic regions, improving reconstruction quality in these areas while reducing quality in static regions, as demonstrated in Figure S3. In scenes with distant background regions, such as *flame\_salmon*, larger  $\gamma$  values result in less densification in the static background, leading to a decline in reconstruction quality. Additionally, excessively high  $\gamma$  values result in over-densification in dynamic regions, increasing the number of anchors. We empirically observe that  $\gamma = 1$  provides a balanced reconstruction between static and dynamic regions.

**Modified temporal opacity.** The parameter  $\beta$  determines the steepness of our generalized Gaussian curve. Higher values of  $\beta$  result in steeper slopes, enabling shapes to be represented with fewer Gaussian components compared to a typical mixture of univariate Gaussians. As shown in Figure S4, increasing  $\beta$  reduces the number of required anchor points while maintaining image quality. However, excessively high  $\beta$  values (e.g., above 8) cause instability during training.

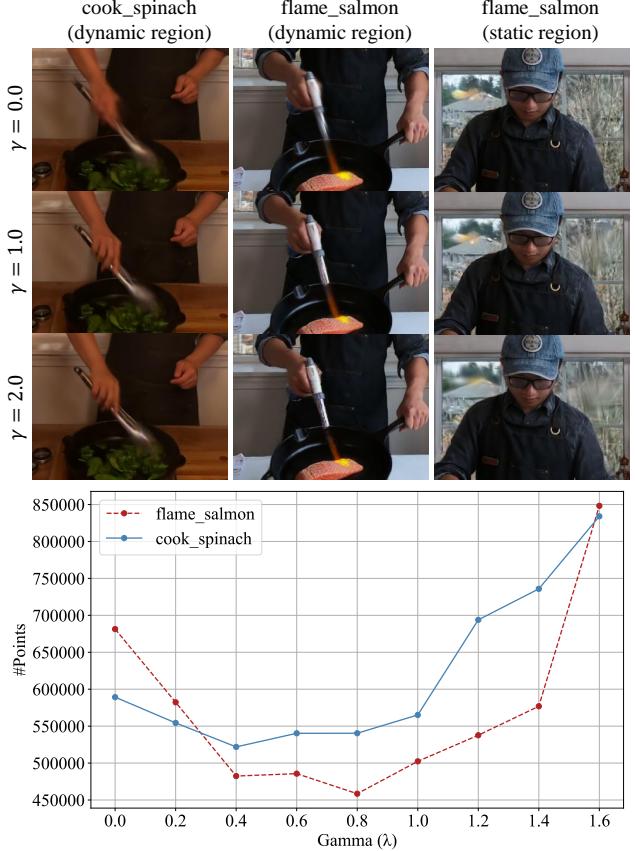


Figure S3. **Effect of  $\gamma$ .** (upper) As the value of  $\gamma$  increases, the quality of dynamic regions improves, whereas the quality of static regions declines. (lower) The number of anchor points changes with  $\gamma$ .

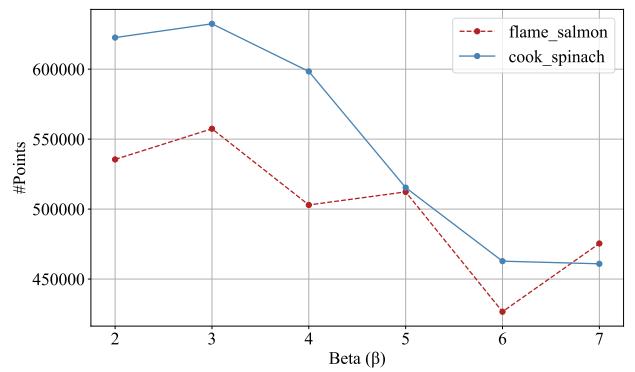


Figure S4. **Effect of  $\beta$ .** The graph shows how  $\beta$  affects the number of anchor points across two scenes, with an increase in  $\beta$  leading to a reduction in the number of anchor points.



Figure S5. **Visualization of Anchor Growing.** The blue dots on the left represent the initial anchor points, which are obtained exclusively at  $t = 0$ . For a specific frame at  $t = 194$ , new anchor points grow as iterations progress, shown as green dots on the right. These new anchor points are primarily generated in under-reconstructed dynamic regions.

## E. Visualization of anchor growing

We visualize the anchor growing process in Figure S5. During the early stages of training, the temporal scale of 4D Gaussians generated from the initial anchors increases, enabling rapid reconstruction of static regions. As training progresses, anchors are created in the dynamic regions at each time step  $t$  in under-reconstructed regions, improving image quality in these areas.

## F. Additional method detail

**Network architecture.** In this section, we describe the detailed architecture of the MLPs utilized in our model. Each anchor feature is represented as a 32-dimensional vector, which is passed through the shared MLPs to compute the properties of  $K$  neural 4D Gaussians. Each MLP consists of a 2-layer architecture with a hidden layer of width 32, matching the size of the feature dimension. The hidden layer uses a ReLU activation function. We employ a total of four MLPs, referred to as the Opacity MLP, Shape MLP, Color MLP, and Velocity MLP.

The Opacity MLP outputs the time-invariant opacity  $\alpha$  of the Gaussians. The Shape MLP produces the quaternion  $\mathbf{q}$  and scaling  $\mathbf{s}$  for covariance calculation, along with the temporal scale  $\sigma$ . The Color MLP takes the direction  $\mathbf{d}$  concatenated with the anchor feature as input and outputs the view-dependent color  $\mathbf{c}$ . The Velocity MLP computes the 3D velocity of the Gaussians.  $\tanh$  for opacity  $\alpha$ ,  $\exp$  for scaling  $\mathbf{s}$  and temporal scale  $\sigma$ , and sigmoid for color  $\mathbf{c}$  are applied as activation function.

**Implementation detail.** The temporal grid size is set to 0.0333 for the N3DV dataset and 0.02 for the Technicolor dataset. For initial points, we downsample it to fewer than 100,000 points. All other hyperparameters, including the learning rate and learning decay schedule, follow those of [16]. Components such as the feature bank, LOD, and appearance features in [16] are excluded.

## G. More results

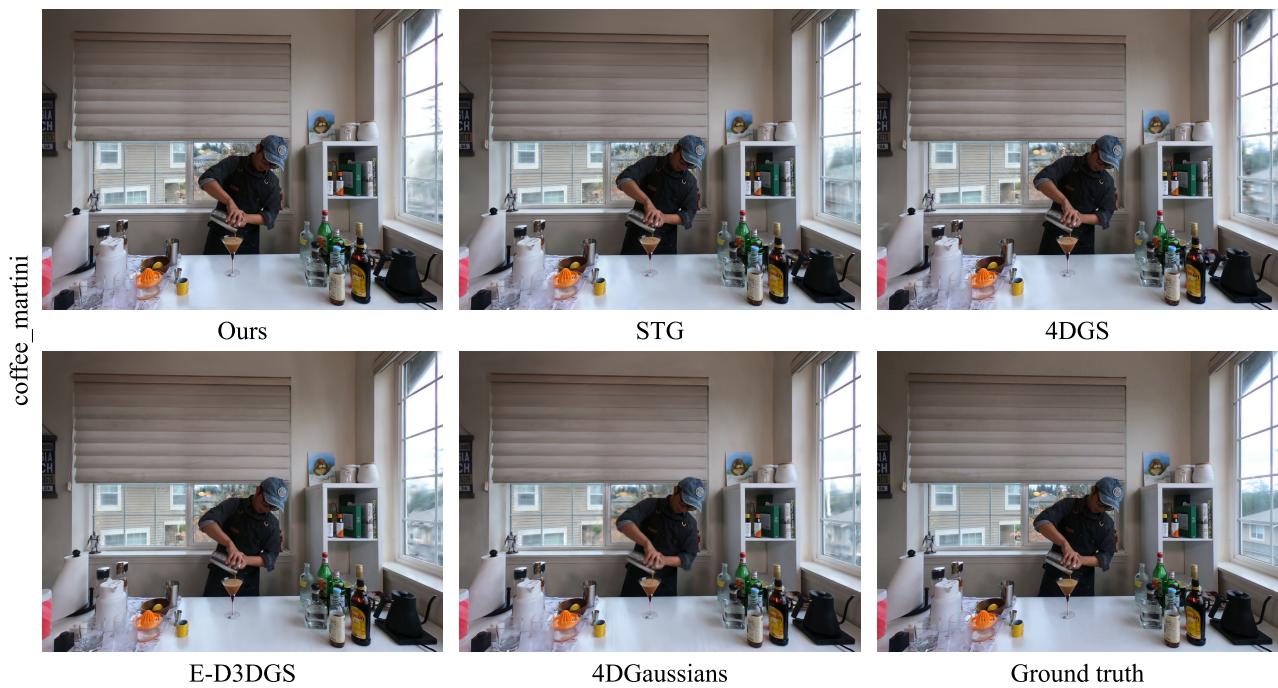
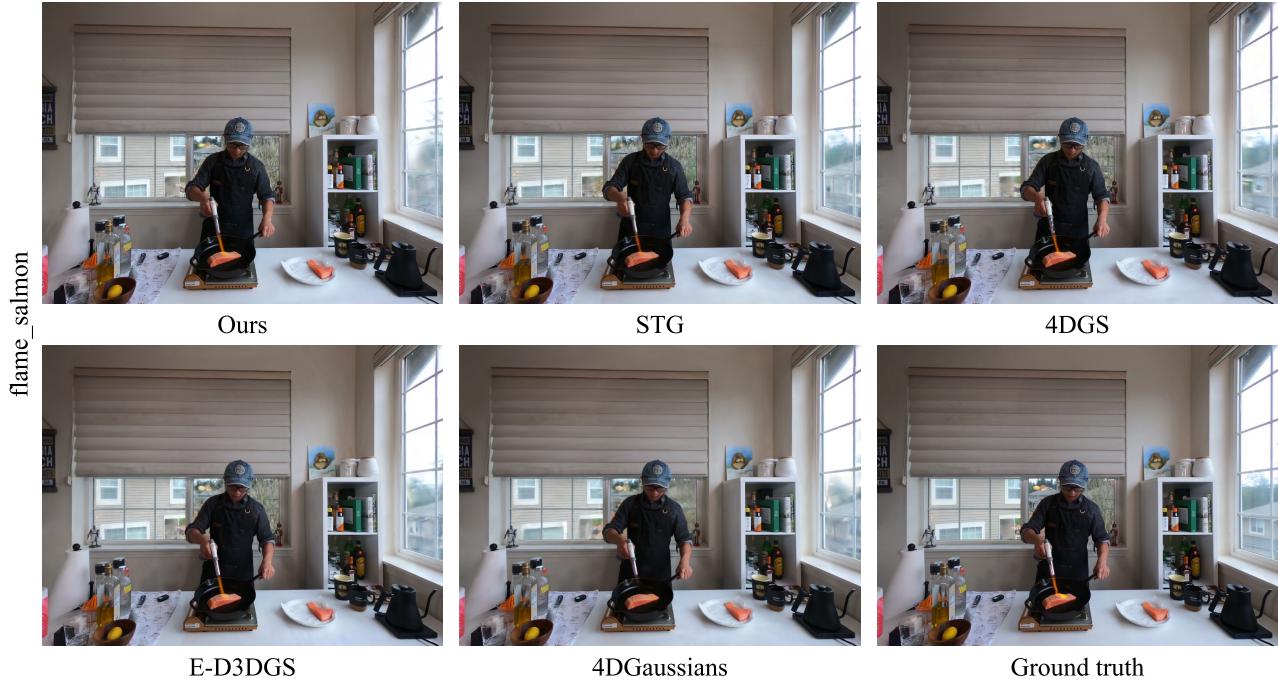
We report the quantitative results in Table S3-S4 and qualitative results in Figure S6-S7 of each scene of the N3DV and the Technicolor datasets, respectively. We also provide video quality comparisons in HTML format. Please check the `index.html` file.

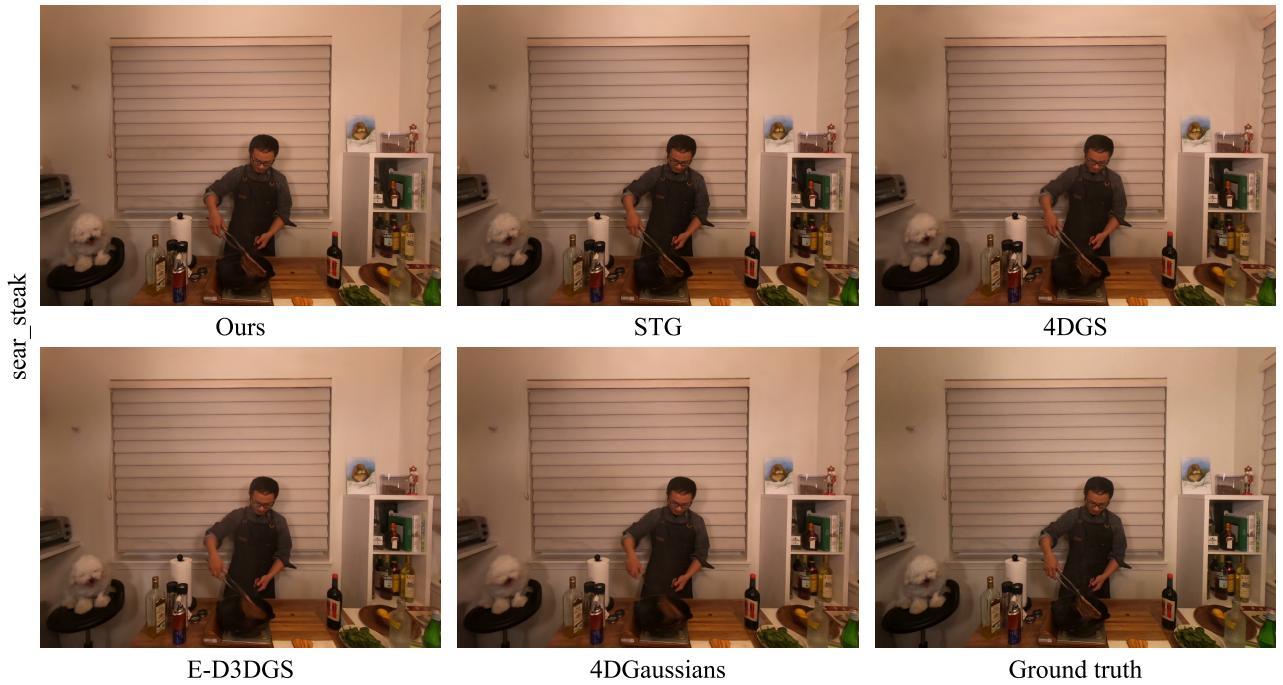
Table S3. Per-scene quantitative results on the N3DV dataset.

Metric \ Model	coffee_martini			cook_spinach			cut_roasted_beef		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
4DGaussians	28.44	0.919	0.060	33.10	0.953	0.042	33.12	0.954	0.044
E-D3DGS	29.10	0.931	0.042	32.96	0.956	0.034	33.57	0.958	0.034
4DGS	28.63	0.918	0.071	33.54	0.956	0.039	34.18	0.959	0.038
STG	28.26	0.915	0.074	33.07	0.958	0.038	33.38	0.959	0.039
<b>Ours</b>	28.80	0.923	0.060	33.36	0.955	0.036	33.35	0.957	0.036
Metric \ Model	flame_salmon			flame_steak			sear_steak		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
4DGaussians	28.80	0.926	0.060	33.55	0.961	0.032	34.02	0.963	0.031
E-D3DGS	29.61	0.936	0.038	33.57	0.964	0.028	33.45	0.963	0.030
4DGS	29.25	0.929	0.057	33.90	0.961	0.037	33.37	0.960	0.040
STG	29.44	0.925	0.067	33.82	0.965	0.031	33.77	0.966	0.031
<b>Ours</b>	29.20	0.928	0.054	33.73	0.960	0.030	33.73	0.960	0.031

Table S4. Per-scene quantitative results on the Technicolor dataset.

Metric \ Model	Birthday			Fabien			Painter		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
4DGaussians	28.03	0.862	0.156	33.36	0.865	0.185	34.52	0.899	0.138
E-D3DGS	33.34	0.951	0.037	34.45	0.875	0.147	36.63	0.923	0.097
4DGS	31.81	0.937	0.040	35.02	0.884	0.144	35.71	0.923	0.104
STG	31.88	0.947	0.028	35.79	0.900	0.115	35.53	0.925	0.095
<b>Ours</b>	32.86	0.952	0.024	35.32	0.889	0.140	37.35	0.941	0.069
Metric \ Model	Theater			Train					
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
4DGaussians	28.67	0.835	0.195	23.54	0.756	0.206			
E-D3DGS	30.64	0.864	0.147	31.84	0.922	0.074			
4DGS	31.94	0.875	0.143	32.25	0.932	0.046			
STG	31.25	0.880	0.133	32.58	0.938	0.039			
<b>Ours</b>	31.65	0.891	0.106	32.69	0.941	0.032			





**Figure S6. Qualitative comparisons on the N3DV dataset.**

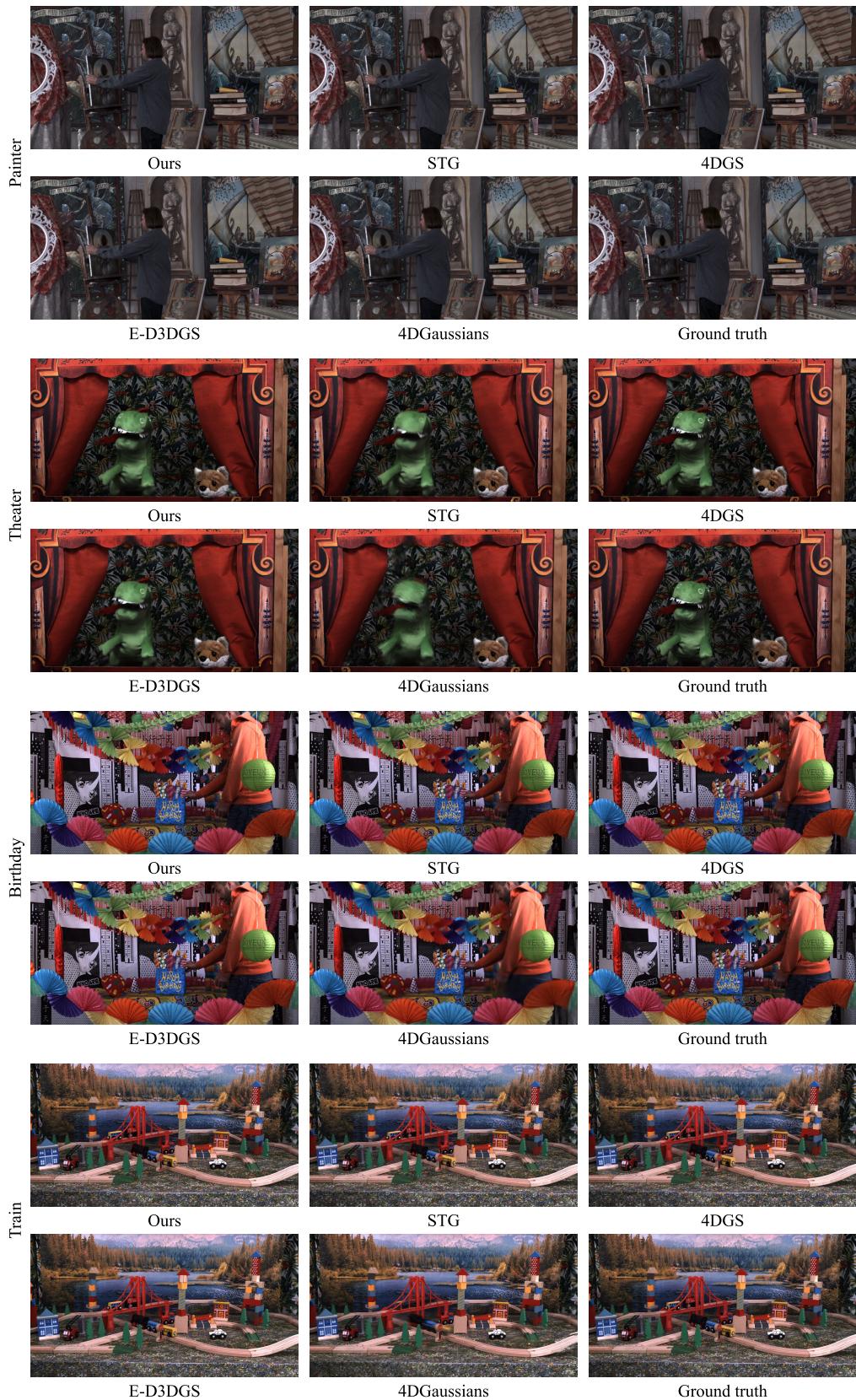


Figure S7. Qualitative comparisons on the Technicolor dataset.