

An Efficient 3D Gaussian Representation for Monocular/Multi-view Dynamic Scenes

Kai Katsumata Duc Minh Vo Hideki Nakayama

The University of Tokyo, Japan

{katsumata, vmduc, nakayama}@nlab.ci.i.u-tokyo.ac.jp

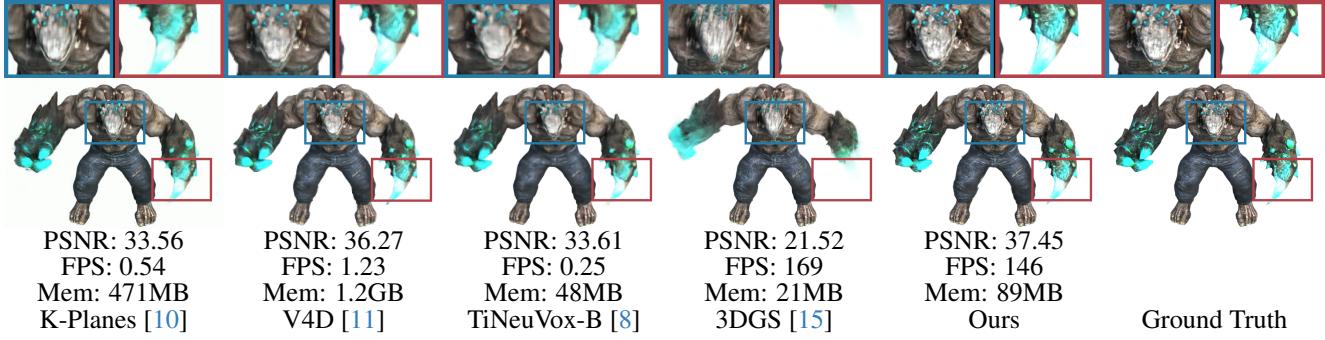


Figure 1. Our method achieves real-time dynamic scene rendering of radiance fields with rendering quality that equals the previous methods with the best quality. Keys to this performance are a memory-efficient dynamic 3D Gaussian representation, which approximates 3D Gaussian parameters changing over time with a few parameters, and the optimization with image and flow reconstruction. We achieve high-quality novel view synthesis of dynamic scenes at a rendering speed slightly slower than 3D Gaussian Splatting [15]. It also suffices with only a minor additional memory consumption. We show the rendering examples on the MUTANT scene in the D-NeRF dataset, visual quality (PSNR), rendering speed (FPS), and memory used to store optimized parameters. Non-obvious differences in quality are highlighted.

Abstract

In novel view synthesis of scenes from multiple input views, 3D Gaussian splatting emerges as a viable alternative to existing radiance field approaches, delivering great visual quality and real-time rendering. While successful in static scenes, the present advancement of 3D Gaussian representation, however, faces challenges in dynamic scenes in terms of memory consumption and the need for numerous observations per time step, due to the onus of storing 3D Gaussian parameters per time step. In this study, we present an efficient 3D Gaussian representation tailored for dynamic scenes in which we define positions and rotations as functions of time while leaving other time-invariant properties of the static 3D Gaussian unchanged. Notably, our representation reduces memory usage, which is consistent regardless of the input sequence length. Additionally, it mitigates the risk of overfitting observed frames by accounting for temporal changes. The optimization of our Gaussian representation based on image and flow reconstruction results in a powerful framework for dynamic scene view syn-

thesis in both monocular and multi-view cases. We obtain the highest rendering speed of 118 frames per second (FPS) at a resolution of 1352×1014 with a single GPU, showing the practical usability and effectiveness of our proposed method in dynamic scene rendering scenarios.

1. Introduction

The landscape of novel view synthesis of scenes captured through multiple images/videos has undergone a revolutionary transformation, owing principally to significant advances in neural radiance field (NeRF) approaches. While achieving remarkable visual quality, particularly in dynamic scenes [12, 19, 26], they inevitably confront hurdles in terms of high-speed training and rendering [22, 24, 25, 27]. This limitation originates from the multi-layer perceptron (MLP) optimization process, leading to lengthy inference times and associated computational costs. The recently proposed 3D Gaussian splatting [15], introduced differentiable 3D Gaussian representation and point-based rasterization, signaling a departure from neural network reliance. No-

tably, 3D Gaussian splatting emerges as a promising solution that not only accelerates both training and rendering processes but also delivers rendered scenes with high quality, rivaling the levels set by NeRF [22]. However, it encounters a challenge in the realm of memory usage and the need for many observations in dynamic scene synthesis [21]. It stores a plethora of 3D Gaussian parameters per each time step. Storing parameters per each time step involves numerous observations per time step, implying the failure in the monocular or few view setups. Our study aims to tackle this issue, paving the way for more memory-efficient while keeping the high-quality rendering characteristic of 3D Gaussian splitting in dynamic scene synthesis.

We introduce an efficient dynamic 3D Gaussian representation, containing time-invariant and time-varying parameters to capture dynamic motion effectively. Like [15, 21], we use scaling factors in the covariance matrix, opacity, and color as time-invariant parameters. Since modeling the change of positions over time is important to represent dynamic scenes [24–26], as time-varying parameters, we express each 3D Gaussian’s position as a function of time to model the temporal change of the position. We fit the position by using the Fourier approximation, inspired by the fact that motion is sometimes periodic, as commonly observed in humans and articulated objects. The time-varying parameters make our representation dynamic, meaning that 3D Gaussian moves and rotates over time. Moreover, as we use a function with a few parameters to represent the position, the small degree of freedom contributes to the smoothness of reconstructed scenes, enhancing the robustness to unseen views. Importantly, our representation’s memory consumption is solely determined by the number of 3D Gaussians and the number of the approximation function’s parameters, independent of the input length. Beyond optimizing Gaussian representations through image-level reconstruction, we further enhance temporal consistency by supervising the Gaussian with optical flow obtained from input videos. This not only ensures high-quality reconstruction but also facilitates the generalization of the representation.

Our experiments on dynamic datasets, including D-NeRF [26], DyNeRF [19], and HyperNeRF [25], demonstrate the effectiveness of optimizing our dynamic 3D Gaussian from both monocular and multi-view videos and show that our proposed method achieves rendering quality that rivals previous neural radiance field approaches [8, 10, 11, 25]. We also can achieve rendering speeds similar to a fast radiance field method [15] and reasonable memory usage, and importantly, our approach offers real-time rendering with high quality for novel view synthesis of dynamic scenes (see Fig. 1). Finally, we show an editing application enabled by the explicit property of 3D Gaussian representations. In a nutshell, our contributions are:

- We present an efficient dynamic 3D Gaussian representa-

tion, which enables real-time novel view synthesis of dynamic scenes from monocular/multi-view video inputs.

- The time-varying parameters introduce a compact representation for modeling dynamic scenes.
- Our dynamic 3D Gaussian representation enables the high-quality real-time dynamic scene rendering of high-resolution images of 1352×1014 at a frame per second (FPS) of 118 with a single GPU.

2. Related Work

We first briefly overview the radiance field for dynamic scenes, then discuss the recent efforts for efficient radiance fields with explicit representation, such as grid-, plane-, hash-, and point-based methods, placing our work in real-time dynamic view synthesis.

2.1. Dynamic view synthesis

Applications in virtual reality and computer vision often require the reconstruction of 4D scenes. Several works extend NeRF [22] to handle dynamic scenes in multi-view or monocular setups by time-varying NeRF [12, 19, 26]. The regularization techniques for temporal smoothness enable suitable scene representations from monocular videos [20]. Additional sensory information is also useful for spatio-temporal regularization. Some attempts [12, 20, 35] employ depth or flow, which are observed or predicted with external networks to reconstruct the scene from sparse observations. Deformation-based approaches [24, 25, 33, 36], another direction of the efforts for dynamic reconstruction, combine static NeRF and deformation fields. Tremendous efforts show high visual quality for dynamic view synthesis. However, they are still afflicted with slow rendering speed. Our study aims to enable real-time dynamic view synthesis with high visual quality. We aim to extend 3D Gaussian splatting to dynamic scene reconstruction to achieve high-speed rendering while maintaining the rendering quality from sparse training views.

2.2. Explicit Radiance Fields

NeRF is an implicit model with a large neural network queried many times during training, making a drawback that it is slow optimization and rendering [22, 38]. A recent line of work [9, 39] addresses the issue in implicit models by exploring explicit models, reducing optimization and rendering time. Plenoxels [9] directly optimizes 3D grid representation instead of neural networks. Generally, explicit models sacrifice visual quality for fast training [9]. Hybrid approaches [6, 10, 23, 32] aim to achieve better trade-offs between training time and visual quality. Instant-NGP allows a compact MLP by exploiting a multi-level hash grid to encode positions to feature vectors [23]. Plane-based approaches are principally designed for representing bounded

scenes [1, 4, 5, 7, 10, 13]. MERF [28] employs a multiresolution representation and a fast contraction function to reconstruct unbounded scenes. For dynamic scenes, K-planes [10] decomposes 4D dynamic volumes into multiple feature planes and employs an MLP-based feature decoder for determining color and density.

2.3. Point-based rendering

Points, which naturally come from depth sensors, Structure from Motion (SfM) [29], or common Multi-View Stereo (MVS) algorithms [30, 31], offer a good representation for representing fine-grained scenes and complex objects, as well as facilitating computationally efficient rendering. Due to these aspects of the representation, they have been well-studied in graphics. The differentiable pipeline for point-based rendering results in that points can be used for reconstructing 3D scenes [15–18]. 3D Gaussian splatting [15], a NeRF alternative, achieves real-time rendering with high visual quality for unbounded static scenes at the expense of generalization performance derived from NeRF’s continuous neural field representation. Recently, Dynamic 3D Gaussians [21] employs 3D Gaussian splatting for dynamic scenes, which models dynamic scenes by the Gaussian position and variance at each timestamp. The position and variance of Gaussians at every timestamp are effective in modeling scenes from dense multi-view dynamic scenes. However, this approach presents difficulties in reconstructing monocular dynamic scenes, resulting in excessive memory consumption, particularly for extended input sequences. We aim to build a memory-efficient Gaussian representation for dynamic scenes, even for monocular scenes, while keeping pure 3D Gaussian representation in order to sacrifice the gift of 3D Gaussians, such as outstanding rendering speed and ease of direct editing of the scene.

3. Overview

The input for our task is a set of images with a timestep and camera parameters obtained from videos. The original 3D Gaussian representation [15] is defined by a position (mean), a covariance matrix (decomposed into a rotation matrix and a scaling vector), an opacity, and a color (determined by spherical harmonics (SH) [3] coefficient). We extend the 3D Gaussian representation for dynamic scenes. Each 3D Gaussian in our method regards the position and rotation in the covariance matrix as time-varying parameters and others as time-invariant parameters over time (Sec. 4). For rendering images, we use the 3D Gaussian splatting technique [15], which renders an image by employing Gaussians within the camera plane out of a set of Gaussians. The technique decides the color for each pixel with α -blending of Gaussians considering their order with respect to the camera view (Sec. 5). For dynamic scene reconstruction, we optimize the Gaussian parameters with the

training frames. Moreover, flow reconstruction enhances the temporal consistency of the learned representation. We render the images from Gaussians, intrinsic and extrinsic camera parameters, and a timestep using a differentiable renderer, and we update the Gaussian parameters to decrease the distance between renderings and training images in the image and flow spaces (Sec. 6). The small degrees of freedom of our representation essentially facilitate the reconstruction of dynamic scenes from a few observations.

4. Dynamic 3D Gaussian representation

To design a compact dynamic 3D Gaussian representation, we aim to express 3D Gaussian parameters using only a few parameters. Our dynamic scene representation comprises of a set of dynamic 3D Gaussians, extending the static 3D Gaussian introduced in [15]. Our dynamic representation lets the 3D Gaussians move through the scene over time, using both time-varying parameters (center position and rotation factors) and time-invariant parameters (scale, color, and opacity). Each dynamic Gaussian encapsulates:

- 1) a 3D position at time t : $[x(t), y(t), z(t)]^\top \in \mathbb{R}^3$,
- 2) a 3D rotation at time t represented by a quaternion: $[q_x(t), q_y(t), q_z(t), q_w(t)]^\top \in \mathbb{R}^4$
- 3) a scaling factor: $[s_x, s_y, s_z]^\top \in \mathbb{R}^3$
- 4) spherical harmonics coefficients representing color with the degrees of freedom k : $h \in \mathbb{R}^{3 \times (k+1)^2}$
- 5) an opacity: $o \in \mathbb{R}$

We approximate the 3D position $x(t), y(t), z(t)$ using Fourier approximation. At time t , it is represented by

$$\begin{aligned} x(t) &= w_{x,0} + \sum_{i=1}^L w_{x,2i-1} \sin(\pi t) + w_{x,2i} \cos(\pi t), \\ y(t) &= w_{y,0} + \sum_{i=1}^L w_{y,2i-1} \sin(\pi t) + w_{y,2i} \cos(\pi t), \\ z(t) &= w_{z,0} + \sum_{i=1}^L w_{z,2i-1} \sin(\pi t) + w_{z,2i} \cos(\pi t), \end{aligned}$$

where, $w_{.,0}, \dots, w_{.,2L}$ are intercept and coefficients of the position, and L is the number of terms (harmonics).

We approximate the 3D rotation (quaternion) over time using a linear approximation. At time t , it is defined as

$$\begin{aligned} q_x(t) &= w_{qx,0} + w_{qx,1}t, \\ q_y(t) &= w_{qy,0} + w_{qy,1}t, \\ q_z(t) &= w_{qz,0} + w_{qz,1}t, \\ q_w(t) &= w_{qw,0} + w_{qw,1}t, \end{aligned}$$

where $w_{.,0}$ and $w_{.,1}$ are intercepts and coefficients of the rotation of Gaussian, respectively.

Each Gaussian at time t is characterized by a 3D covariance matrix $\Sigma(t)$ and a 3D center $\mu(t) = [x(t), y(t), z(t)]^\top$. The density of 3D Gaussian on a ray \vec{x} is obtained as follows:

$$G_t(\vec{x}) = e^{-\frac{1}{2}(\vec{x}-\mu(t))^\top \Sigma(t)^{-1}(\vec{x}-\mu(t))}.$$

To constrain the covariance matrix $\Sigma(t)$ to be a positive semi-definite matrix during optimization, the covariance matrix $\Sigma(t)$ is decomposed by using a scaling matrix $\mathbf{S} = \text{diag}(s_x, s_y, s_z)$ and a rotation matrix $\mathbf{R}(t)$ as

$$\Sigma(t) = \mathbf{R}(t)\mathbf{S}\mathbf{S}^\top\mathbf{R}(t)^\top.$$

Here, a rotation matrix $\mathbf{R}(t)$ is represented by quaternion $(q_x(t), q_y(t), q_z(t), q_w(t))$.

For each Gaussian, the preceding definition yields $3L + 8 + 3 + 3(k+1)^2 + 1$ parameters with respect to 3D center, 3D rotation, scale, color, and opacity. Notably, the parameter count for each Gaussian is defined merely by the number of approximation terms and spherical harmonic degrees of freedom, with no regard to time length. When compared to approaches that require parameters for each time step, this approach saves on memory usage. Memory consumption in our dynamic scene representation is determined by two hyperparameters (*i.e.*, L and k) and the number of Gaussians used. Furthermore, the representation, which is defined as a function of time over continuous time, inhibits discontinuous movement through time. This characteristic improves robustness in novel view synthesis settings.

5. Rendering via 3D Gaussian Splatting

Rendering with 3D Gaussian applies splatting techniques [15] to the Gaussian within the camera planes. Zwicker et al. [40] give the projection of the 3D covariance matrix to 2D covariance matrix. The 3D covariance matrix Σ is projected into 2D covariance matrix Σ' given a viewing transformation \mathbf{W} as follow:

$$\Sigma'(t) = \mathbf{J}\mathbf{W}\Sigma(t)\mathbf{W}^\top\mathbf{J}^\top,$$

where \mathbf{J} is the Jacobian of the affine approximation of the projective transformation at Gaussian center $\mu(t)$:

$$\mathbf{J} = \begin{bmatrix} \frac{1}{v_z} & 0 & -\frac{v_x}{v_z^2} \\ 0 & \frac{1}{v_z} & -\frac{v_y}{v_z^2} \\ 0 & 0 & 0 \end{bmatrix}, \quad (1)$$

where $[v_x, v_y, v_z]^\top = \mathbf{W}\mu(t)$ is the camera coordinate of the Gaussian center $\mu(t)$ obtained by the viewing transformation, which projects the points from the world space to the camera space.

Similar to NeRF style volumetric rendering, the point-based rendering computes the color C of a pixel by evaluating the blending of N ordered points that overlap the pixel:

$$C = \sum_{i=1}^N c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j),$$

where, c_i represents the color of a Gaussian evaluated by SH coefficients, and α_i represents the density that is calculated from a 2D Gaussian with covariance Σ' at time t and a 2D center μ' at time t and an optimized opacity o .

6. Optimization of the dynamic 3D Gaussian representation

We optimize the Gaussian parameters: intercepts and coefficients of position and rotation w , a scaling factor s_x, s_y, s_z , SH coefficients h and an opacity o , based on the iterations of rendering and comparing the rendered images with training frames in the captured videos. To compare the rendered and training views, the loss function contains L1 and D-SSIM:

$$\mathcal{L}_{\text{recon}} = (1 - \lambda)|\hat{I} - I| + \lambda \mathcal{L}_{D-SSIM},$$

where I and \hat{I} are target and rendered images, respectively. The loss function moves and rotates the anisotropic Gaussians and changes their color and opacity so that each Gaussian covers a homogeneous area. Since it just fixes incorrectly positioned Gaussians, the over- or under-representation of the set of Gaussians for the scene needs a mechanism for creating Gaussians that reconstruct the scene or destroy extra Gaussians. We also follow the divide and prune techniques in 3D Gaussian splatting for producing a compact and precise representation of the scene. We surveil the gradients of each Gaussian and densify Gaussians by splitting a Gaussian with a large gradient and a large scale into two small Gaussians and cloning a Gaussian with a large gradient and a small scale to two Gaussians. Moreover, we remove transparent Gaussians with an opacity less than a threshold value of 0.005.

Earning the prior of the point sets for a dynamic scene to gain in the reconstruction of the scene is challenging. Instead of the initialization of a set of Gaussians with a set of sparse points from SfM [29], we initialize a set of Gaussians randomly using a uniform distribution. It avoids time-intensive SfM and challenges in SfM from dynamic scenes, in particular a monocular setting. Deeming the frames in the captured datasets as static scene, and we optimize static representation in the former static stage to learn the prior of Gaussians. In other words, we optimize the parameters consistent all over time (*i.e.*, scale, SH coefficients, and opacity) and intercepts for a center and a rotation ($w_{x,0}, w_{y,0}, w_{z,0}, w_{qx,0}, w_{qy,0}, w_{qz,0}, w_{qw,0}$) out of the Gaussian parameters, in the static stage. After the static

stage, we optimize the whole parameters of the set of Gaussians to reconstruct a dynamic region as a dynamic stage.

Another challenge in the reconstruction of the dynamic scene is ambiguity come from the limited number of captured views at a timestep. Since a dynamic scene contains temporal changes, such as moving objects and changing shape, it is hard to share the scene information over frames with different timesteps. To overcome the ambiguity, we employ flow information. We supervises the flows of the optimizable Gaussians with the ground truth optical flows of the input frames. We use a RAFT algorithm [34] to obtain ground truth flow for training views: forward flow f_{fwd} and backward flow f_{bwd} between two adjacent frames. We combine the flow loss $\mathcal{L}_{\text{flow}}$ with the reconstruction loss that compare the renderings and training views:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda_{\text{flow}} \mathcal{L}_{\text{flow}}(\hat{F}, F)$$

where, $F = \{f_{\text{fwd}}, f_{\text{bwd}}\}$ and \hat{F} are the ground truth flow and the flow of the Gaussians, respectively, and λ_{flow} is a balancing hyperparameter for the flow term. Instead of applying an optical flow algorithm for renderings, we create pseudo optical flow from Gaussian representation. The motion of the scene is represented by only the coefficients of the 3D Gaussian mean $w_{x,1 \leq i}, w_{y,1 \leq i}, w_{z,1 \leq i}$. We can calculate the scene flow in the 3D space by

$$\begin{aligned}\hat{f}_{\text{fwd}}^x &= x(t + \Delta t) - x(t), & \hat{f}_{\text{fwd}}^x &= x(t) - x(t - \Delta t), \\ \hat{f}_{\text{fwd}}^y &= y(t + \Delta t) - y(t), & \hat{f}_{\text{fwd}}^y &= y(t) - y(t - \Delta t), \\ \hat{f}_{\text{fwd}}^z &= z(t + \Delta t) - z(t), & \hat{f}_{\text{fwd}}^z &= z(t) - z(t - \Delta t),\end{aligned}$$

where Δt is the difference between the timesteps of the two image frames. The scene flow is projected into 2D camera plane using

$$\hat{f}_{\{\text{fwd,bwd}\}}^{\text{xyz}} = \mathbf{J}[\hat{f}_{\{\text{fwd,bwd}\}}^x, \hat{f}_{\{\text{fwd,bwd}\}}^y, \hat{f}_{\{\text{fwd,bwd}\}}^z]^T,$$

where \mathbf{J} is the Jacobian of the affine approximation of the projective transformation at the Gaussian center μ (Eq. (1)). Regarding the scene flows on camera plane as RGB colors, we can apply the point-based rendering for calculate an optical flow of a pixel as the α -blending:

$$\hat{f}_{\text{fwd}} = \sum_{i=1}^N \hat{f}_{\text{fwd},i}^{\text{xyz}} \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j).$$

The backward flow is calculated in the same way. The optical flow \hat{F} consists of the forward flows \hat{f}_{fwd} and backward flows \hat{f}_{bwd} for the all pixels. The flow loss $\mathcal{L}_{\text{flow}}$ takes L1 loss between the ground truth flows and the optical flow of the Gaussian for both directions of the flows. The flow loss provides the spatial-temporal consistency to our method without the additional computation cost in rendering since it does not need additional components. We

Table 1. Quantitative results on the D-NeRF dataset [26]. Our method performs competitively against neural radiance field approaches in terms of visual quality and achieves fastest rendering speed in the previous methods with the high quality. Results excepting FPS of [8, 10, 11] are adopted from the original papers.

	PSNR↑	MS-SSIM ↑	FPS ↑	Train Time ↓	Mem↓
TiNeuVox-S [8]	30.75	0.96	0.32	8 mins	8MB
TiNeuVox-B [8]	32.67	0.97	0.13	28 mins	48MB
K-Planes [10]	31.61	0.97	0.54	52 mins	~497MB
V4D [11]	33.72	0.98	1.47	6.9 hrs	1.2GB
3DGs [15]	20.51	0.89	170	6 mins	~50MB
D-3DGs	17.22	0.81	173	15 mins	~913MB
Ours	32.07	0.96	150	8 mins	~159MB

Table 2. Quantitative results on the DyNeRF scenes [19].

	PSNR↑	MS-SSIM ↑	FPS ↑	Train Time ↓	Mem↓
K-Planes [10]	31.63	0.964	0.31	1.8 hrs	~309MB
3DGs [15]	20.94	0.800	109	20 mins	~198MB
D-3DGs	24.36	0.834	119	51 mins	~2.3GB
Ours	28.89	0.945	118	1 hrs	~338MB

exclude the flow loss for the D-NeRF dataset because the teleport of the cameras between neighbouring frames causes difficulties in calculating ground truth flows.

7. Experiment

7.1. Evaluation data

We evaluate our dynamic Gaussian presentation using three dynamic scene datasets: a synthetic one D-NeRF [26], and two real ones DyNeRF [19] and HyperNeRF [25].

D-NeRF dataset [26]. This dataset comprises eight videos of varying lengths, ranging from 50 to 200 frames per video. The camera setup is designed to mimic a monocular camera setting by teleporting between adjacent time steps. The test views are from novel camera positions. We train and render at the resolution of 800×800 .

DyNeRF dataset [19]. The multi-camera dataset includes six 10-second videos captured at 30 FPS using 15 – 20 synchronized fixed cameras. For evaluation, a central camera is used, while training utilizes frames from the other cameras. The training and rendering resolution is set at 1352×1014 .

HyperNeRF dataset [25]. This dataset encompasses videos ranging from 8 to 15 seconds, captured at 15 FPS using two Pixel 3 phones. The training and rendering processes are conducted at a resolution of 540×960 .

7.2. Implementation details

We adhere to the experimental setup in the 3D Gaussian splatting paper [15]. The number of approximation terms of Gaussian centers L is set to 2 for the D-NeRF dataset and 5 for both the DyNeRF and HyperNeRF datasets. Our two-stage optimization process begins with an initial fitting

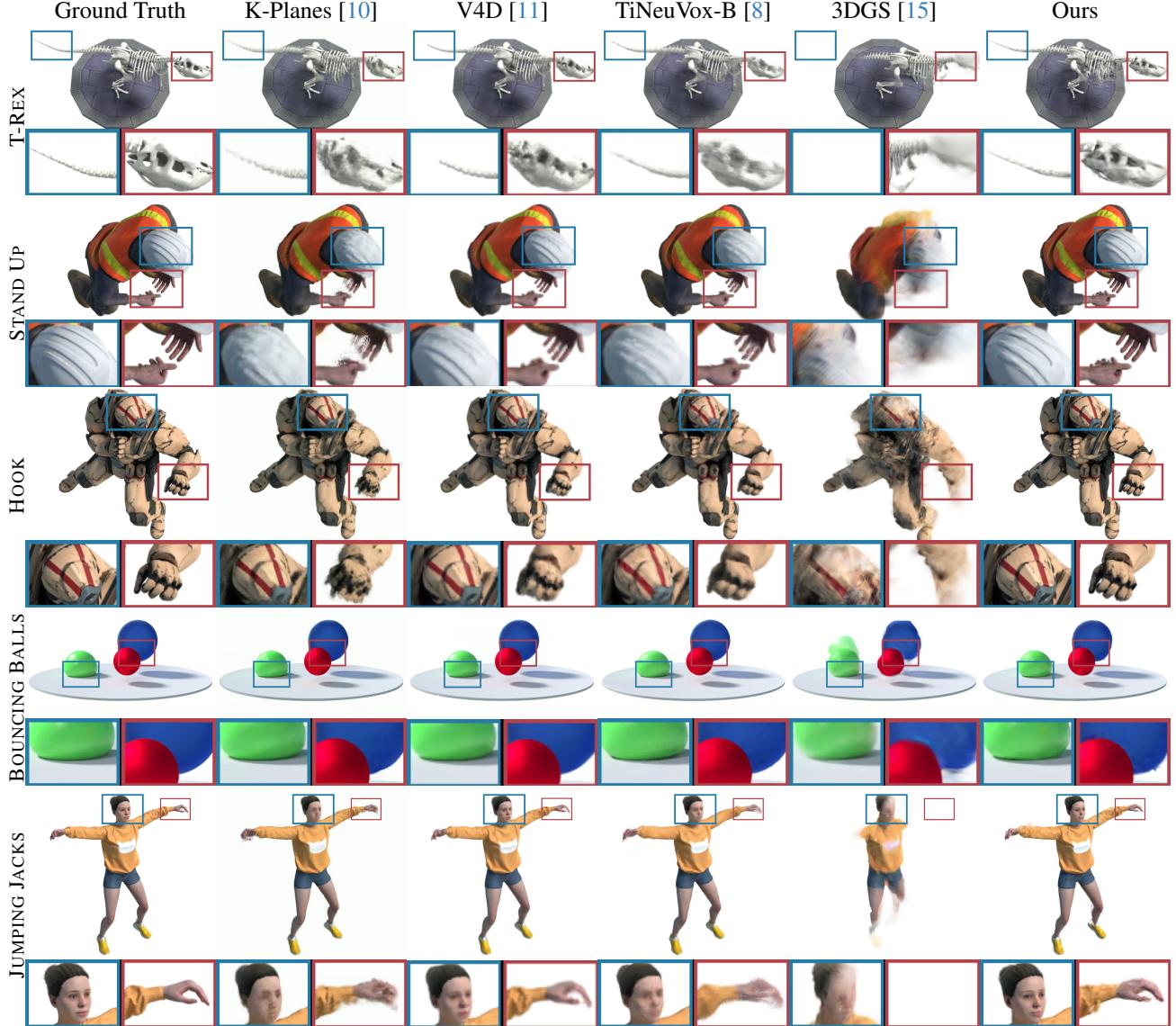


Figure 2. Qualitative comparison on D-NeRF [26]. We highlight the difference by zoom view. Our method achieves competitive visual quality with strong baselines. While our method successfully reconstructs intricate details like hands, it causes the blurred sphere shape.

of parameters, excluding the coefficients for Gaussian position and variance. This initial stage spans 3000 iterations and utilizes all training views in a static setting. Subsequently, we engage in a dynamic stage, adjusting all Gaussian parameters in 27000 iterations. The entire optimization process encompasses 30000 iterations in total. We set the flow loss weight λ_{flow} at 1000 and acquire ground truth flow through RAFT pretrained on the Sintel dataset [2]. All experiments are conducted on a single RTX A6000 GPU.

7.3. Evaluation setup

Compared methods. We benchmark our method against several SoTA methods, including TiNeuVox [8], K-

Planes [10], V4D [11], HyperNeRF [25], 3D Gaussian Splatting (3DGS) [15], and a D-3DGS baseline. Notably, D-3DGS is a dynamic extension of the 3DGS method, which stores both position and rotation for each time step.

Evaluation metrics. We assess the methods using various metrics, including PSNR, SSIM [37], FPS, Training time, and memory used to store optimized parameters. Memory consumption includes 3D Gaussian parameters, neural network parameters, voxel/plane representation, and so on.

7.4. Results

Quantitative results. The quantitative results on the D-NeRF dataset are detailed in Tab. 1. Our method demon-

Table 3. Quantitative results on the HyperNeRF dataset [25].

	FPS↑	Train Time↓	Mem↓	BROOM		3D PRINTER		CHICKEN		PEEL BANANA		Mean	
				PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
HyperNeRF [25]	0.36	48 hrs†	15MB	19.3	0.591	20.0	0.821	26.9	0.948	23.3	0.896	22.2	0.811
TiNeuVox-B [8]	0.14	30 mins	48MB	21.5	0.686	22.8	0.841	28.3	0.947	24.4	0.873	24.3	0.837
V4D [11]	0.15	7 hrs	1.2GB	22.1	0.669	23.2	0.835	28.4	0.929	25.2	0.873	24.7	0.827
Ours	188	1 hrs	~720MB	20.4	0.660	21.1	0.754	25.8	0.859	26.6	0.920	23.5	0.798

†Train time of HyperNeRF [25] is estimated from descriptions in their paper. Originally reported as 8 hours on 4 TPU v4s [14], the TPU v4 is slightly faster than the A100 GPU, and the A100 GPU is at least 1.5 times faster than the A6000 GPU.



Figure 3. Qualitative comparison on DyNeRF [19]. We highlight the difference by zoom view.

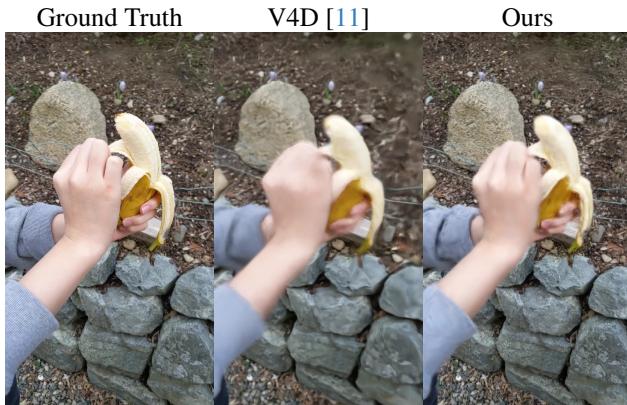


Figure 4. Qualitative comparison on HyperNeRF [25]. Our method offers sharp results.

strates performance comparable to TiNeuVox and K-Planes in terms of visual quality indicated by PSNR and SSIM. Notably, it excels in training time, FPS, and memory size, boasting a rendering speed that is $300\times$ faster than K-Planes. Furthermore, our method surpasses both 3DGS and D-3DGS in terms of visual quality without compromising rendering speed. In the DyNeRF scenes experiment,

Table 4. Per-scene quantitative comparison on D-NeRF scenes of different L . The highest mean score is achieved with $L = 2$, but increasing the complexity L improves visual quality in some scenes (JUMPING JACKS and T-REX).

	PSNR								
	STAND UP	JACKS	BALLS	LEGO	WARRIOR	HOOK	T-REX	MUTANT	Mean
K-Planes [10]	33.72	32.64	41.22	25.47	25.70	28.50	31.79	33.56	31.61
$L = 1$	40.21	27.22	30.27	24.00	32.42	32.84	25.15	38.04	31.26
$L = 2$	39.10	30.95	33.29	22.21	34.15	33.19	26.22	37.45	32.07
$L = 3$	38.09	32.78	32.54	19.58	35.36	30.23	27.73	35.06	31.42
$L = 4$	35.83	32.93	30.39	18.17	34.38	27.74	28.17	32.58	30.02
$L = 5$	32.89	30.71	27.68	17.63	32.64	25.43	26.11	29.09	27.77
MS-SSIM									
K-Planes [10]	0.983	0.977	0.992	0.948	0.952	0.954	0.981	0.983	0.971
$L = 1$	0.989	0.954	0.979	0.899	0.937	0.967	0.960	0.986	0.958
$L = 2$	0.987	0.970	0.983	0.883	0.948	0.967	0.964	0.982	0.960
$L = 3$	0.984	0.975	0.982	0.837	0.955	0.951	0.967	0.971	0.952
$L = 4$	0.977	0.973	0.975	0.810	0.947	0.929	0.966	0.957	0.941
$L = 5$	0.965	0.964	0.966	0.795	0.933	0.904	0.950	0.931	0.926

detailed in Tab. 2, while our method does not exceed the baseline in reconstruction quality, it shows a substantial improvement in FPS. Since the DyNeRF scenes contain multi-view data, the D-3DGS baseline naturally improves static 3DGS, unlike monocular scenes. Impressively, our method even attains rendering speeds that exceed real-time performance, at a high resolution of 1354×1014 . In the challenging HyperNeRF dataset, which is captured by only two moving cameras, referenced in Tab. 3, our method not only



Figure 5. Qualitative comparison of disabled and enabled flow loss on DyNeRF. We highlight the difference by zoom view.



Figure 6. Composition of two scenes. Our method allows adding 3D objects represented 3D Gaussian into a 3D Gaussian scene. We highlight the added object.

demonstrates rapid rendering speeds but also attains an average PSNR that exceeds that of HyperNeRF. It excels particularly in the PEEL BANANA scene, achieving the highest PSNR and MS-SSIM scores. However, our method’s robustness is compromised in scenarios with inaccurate camera poses (3D PRINTER). Our explicit representation and the straightforward rasterization do not yield plausible results for contradictory views.

Qualitative results. Figs. 2 to 4 show that our method yields sharper rendering results and it frequently reconstructs fine details of the scenes accurately. Nonetheless, some cases lacking the approximation complexity lead to blurred results, such as T-REX scene in D-NeRF (Fig. 2), and “steak” in the DyNeRF scene (Fig. 3).

Effect of the number of parameters L . Tab. 4 shows per-scene PSNR and SSIM scores of K-Planes and our method with the different L . It is observed that the optimal L for novel view synthesis varies from scene to scene, highlighting the necessity for complex approximations to capture intricate motions effectively. Additionally, visual comparisons drawn from our method with and without the flow loss (Fig. 5) reveal that incorporating the flow loss mitigates ghostly artifacts and significantly enhances the accuracy of color reconstruction.

Scene composition. Since our dynamic 3D Gaussian representation still uses pure 3D Gaussian representation, the

learned representation is easy to edit Gaussians. We demonstrate the composition of two scenes with our representation. Fig. 6 illustrates this by combining the MUTANT scene from the D-NeRF dataset with the SEARED STEAK scene from the DyNeRF dataset. This demonstrates our method’s capability in editing dynamic 3D scenes.

8. Discussion and Conclusion

Limitations and future directions. Our dynamic Gaussians is defined over all time of the dynamic scene. This representation implicitly assumes Gaussians exist over all time of the scene. It enables us to model naturally the rigid and non-rigid deformation in the scene. On the other hand, it is hard to model the change of topology, the occurrence of Gaussians, and extinction of Gaussians (*e.g.*, fluid). To overcome the limitation, considering the lifetime of Gaussians, such as adding start and end time parameters, will model changes in scene topology.

Our Gaussian representation sacrifices the continuity and smoothness of neural field-based volume rendering, causing performance drops in inaccurate camera poses and degradation of generalization performance. A potential extension of our method is the distillation of the learned NeRF model, which improves the rendering quality without losing the advantage of fast rendering of our method.

Conclusion. We present a dynamic 3D Gaussian representation that allows the accurate reconstruction of dynamic scenes and real-time rendering. We propose a representation for position and variance of 3D Gaussians as a function of time for modeling the motion of the scene. Our model reconstructs dynamic scenes by fitting the Gaussian parameters using the views of all timesteps. Furthermore, we introduce the flow loss that constrains the scene flow of learned Gaussian representation with the ground truth flow. Our experiments on synthetic and real datasets show that the proposed method achieves real-time dynamic scene rendering even for a high resolution. We will explore the rendering with higher rendering quality with the aid of recent efforts on dynamic scene reconstruction.

References

- [1] Sizhe An, Hongyi Xu, Yichun Shi, Guoxian Song, Umit Y Ogras, and Linjie Luo. PanoHead: Geometry-aware 3D full-head synthesis in 360deg. In *CVPR*, pages 20950–20959, 2023. 3
- [2] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *ECCV*, pages 611–625, 2012. 6
- [3] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. *SIGGRAPH*, 21(4):273–281, 1987. 3
- [4] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. In *CVPR*, pages 130–141, 2023. 3
- [5] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3D generative adversarial networks. In *CVPR*, pages 16123–16133, 2022. 3
- [6] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *ECCV*, pages 333–350, 2022. 2
- [7] Zijian Dong, Xu Chen, Jinlong Yang, Michael J Black, Otmar Hilliges, and Andreas Geiger. AG3D: Learning to generate 3D avatars from 2D image collections. In *ICCV*, pages 14916–14927, 2023. 3
- [8] 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. 1, 2, 5, 6, 7
- [9] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, pages 5501–5510, 2022. 2
- [10] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-Planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, pages 12479–12488, 2023. 1, 2, 3, 5, 6, 7
- [11] Wanshui Gan, Hongbin Xu, Yi Huang, Shifeng Chen, and Naoto Yokoya. V4d: Voxel for 4d novel view synthesis. *IEEE TVCG*, 2023. 1, 2, 5, 6, 7
- [12] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *ICCV*, pages 5712–5721, 2021. 1, 2
- [13] Honglin He, Zhuoqian Yang, Shikai Li, Bo Dai, and Wayne Wu. OrthoPlanes: A novel representation for better 3D-awareness of GANs. In *ICCV*, pages 22996–23007, 2023. 3
- [14] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–14, 2023. 7
- [15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM TOG*, 42(4):1–14, 2023. 1, 2, 3, 4, 5, 6
- [16] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *ECCV*, pages 596–614, 2022.
- [17] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Comput. Graph. Forum*, pages 29–43, 2021.
- [18] Georgios Kopanas, Thomas Leimkühler, Gilles Rainer, Clément Jambon, and George Drettakis. Neural point catacaustics for novel-view synthesis of reflections. *ACM TOG*, 41(6):1–15, 2022. 3
- [19] 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 *CVPR*, pages 5521–5531, 2022. 1, 2, 5, 7
- [20] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *CVPR*, pages 6498–6508, 2021. 2
- [21] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024. 2, 3
- [22] 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. 1, 2
- [23] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM TOG*, 41(4):1–15, 2022. 2
- [24] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *ICCV*, pages 5865–5874, 2021. 1, 2
- [25] 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 TOG*, 40(6):1–12, 2021. 1, 2, 5, 6, 7
- [26] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *CVPR*, pages 10318–10327, 2021. 1, 2, 5, 6
- [27] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, pages 14335–14345, 2021. 1
- [28] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. MERF: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM TOG*, 42(4):1–12, 2023. 3
- [29] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 3, 4

- [30] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*, pages 501–518, 2016. 3
- [31] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*, pages 519–528, 2006. 3
- [32] 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 *CVPR*, pages 16632–16642, 2023. 2
- [33] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. NeRF-Player: A streamable dynamic scene representation with decomposed neural radiance fields. *IEEE TVCG*, 29(5):2732–2742, 2023. 2
- [34] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, pages 402–419, 2020. 5
- [35] Fengrui Tian, Shaoyi Du, and Yueqi Duan. Monon erf: Learning a generalizable dynamic radiance field from monocular videos. In *ICCV*, pages 17903–17913, 2023. 2
- [36] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhofer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *ICCV*, 2021. 2
- [37] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE TIP*, 13(4):600–612, 2004. 6
- [38] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Comput. Graph. Forum*, 2022. 2
- [39] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, pages 5752–5761, 2021. 2
- [40] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. EWA splatting. *IEEE TVCG*, 8(3):223–238, 2002. 4