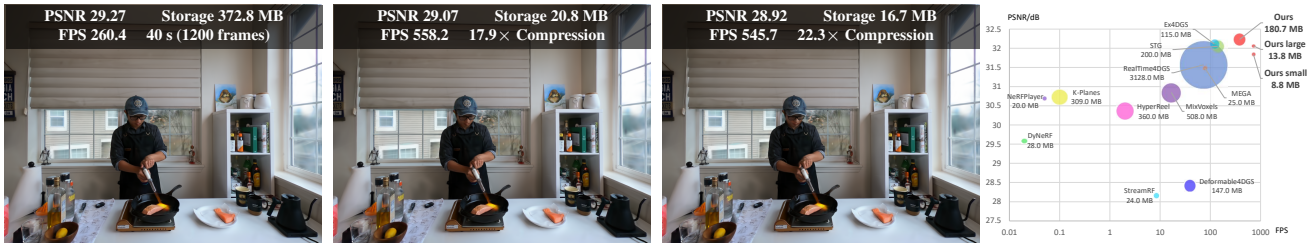
Layered 4D-Rotor Gaussian Splatting: A Compressed Representation for Long Dynamic Scenes

Hanjie Xu[1*] Yuanxing Duan[2*] Qiyu Dai[3*] Ge Li[1†] Baoquan Chen[3†] He Wang[2,4†]

[1]School of Electronic and Computer Engineering, Peking University [2]Galbot
[3]School of Intelligence Science and Technology, Peking University
[4]Center on Frontiers of Computing Studies, Peking University



Ours Ours Large Ours Small PSNR *vs*. FPS *vs*. Storage

Figure 1. We present Layered 4D-Rotor Gaussian Splatting, a novel compressed representation for long dynamic scenes. Our base method (**Ours**) addresses the challenge of long-video modeling, and two further compression variants (**Ours Large, Ours Small**) significantly reduce storage while preserving fine details and improving rendering speed. The proposed method outperforms previous works in rendering quality with minimal storage, and achieves a rendering speed of 662 FPS on the N3DV dataset [17] using an NVIDIA RTX 3090 GPU.

Abstract

We consider the problem of reconstructing long dynamic scenes from multi-view videos in a storage-efficient manner. Recent advances in Gaussian Splatting and its extensions to dynamic scenes have demonstrated impressive visual quality, but remain limited to short duration (<10 s), large storage size (>500 MB), and high GPU VRAM usage. To overcome these limitations, we introduce Layered 4D-Rotor Gaussian Splatting (L4DRotorGS), a novel compressed representation designed for long dynamic scenes. Our approach integrates a layered 4D representation, efficient training, and effective compression into a unified framework. Specifically, 4D Gaussians are first organized into layers based on their temporal extents and then partitioned into discrete temporal buckets. This structure allows for selective access and rendering of only the necessary subsets of 4D Gaussians, substantially reducing GPU memory requirements. To further compress the representation, we apply a series of techniques, Factorized Covariance Quantization, Layered Compression, and Residual Codebook Quantization, achieving a compression ratio of up to 22.3× while preserving high visual fidelity. We implement a highly optimized C++/CUDA framework for efficient training, compression, and real-time rendering, achieving over 500 FPS on an RTX 3090 GPU. Extensive experiments demonstrate the superior storage efficiency, visual quality, and rendering speed of L4DRotorGS, consistently outperforming prior methods in both quantitative metrics and perceptual quality on real-world long dynamic scenes. Project page can be found at https://m1sak1-mei.github.io/layered-4d-rotor/.

1. Introduction

Reconstructing dynamic scenes from multi-view videos and synthesizing novel views is a fundamental yet challenging problem in computer vision and computer graphics. This task is critical for a wide range of applications, including AR/VR, film production, and interactive entertainment. While significant progress has been made in novel view synthesis (NVS) for static scenes, extending these techniques to dynamic environments remains an under-explored problem. The core difficulty lies in accurately modeling complex spatio-temporal deformations and diverse motion patterns across time and viewpoints. These challenges are further compounded by the need for high visual fidelity, real-time rendering, and efficient storage usage, especially when

*Joint first authors.
†Corresponding authors.

handling long video sequences.

Prior efforts [1, 2, 17, 26, 31] have attempted to tackle dynamic NVS using Neural Radiance Fields (NeRF)-based methods, which typically learn deformation fields from a canonical space to represent time-varying geometries and appearances. However, these approaches often suffer from slow volume rendering, making them impractical for real-time or long-duration applications. The recent advent of 3D Gaussian Splatting (3DGS) has significantly advanced the field by enabling fast, high-fidelity rendering through efficient tile-based rasterization. Building on this, several works [4, 36] have extended 3D Gaussians to the 4D domain, modeling dynamic scenes as anisotropic 4D *XYZT* Gaussians. By slicing the temporal dimension, these 4D Gaussians produce time-specific 3D Gaussians that can be efficiently projected to 2D for rendering. Despite their impressive performance on short video sequences, these approaches scale poorly to long video content due to the rapid growth in the number of Gaussians, which leads to large GPU VRAM and storage requirements. More recently, Temporal Gaussian Hierarchy (TGH) [34] explores the extension of 4D Gaussians to long videos, but their method fails to meet low bandwidth requirements (*e.g.*, < 1 MB/s), which are critical for practical deployment in bandwidth-constrained environments such as mobile platforms.

In this paper, we propose a novel framework, Layered 4D-Rotor Gaussian Splatting (L4DRotorGS), that models long-duration dynamic scenes with compressed representations, enabling high-fidelity reconstruction and real-time rendering, while significantly reducing storage costs. Our method includes a layered 4D Gaussian representation, an efficient training framework, and effective compression techniques.

To construct the layered 4D Gaussian representation, we draw inspiration from TGH, which organizes Gaussians into a multi-level hierarchy based on their temporal extents. While adopting a similar strategy, we introduce a more compression-friendly design tailored for long dynamic sequences. Specifically, we further partition each TGH segment into left and right temporal buckets, while allowing Gaussians to span across bucket boundaries. This leads to more temporally consistent Gaussian groups within each layer, facilitating downstream compression.

For the training framework, we introduce a triple-buffer strategy and a streamlined training pipeline that minimizes memory transfer overhead between the CPU and GPU. In addition, we propose a Dynamic-Aware Rotor Learning Rate (DARLR) strategy that enhances training stability for static regions within long dynamic scenes.

Finally, to compress this layered 4D representation, we develop a series of techniques specifically designed for its layered structure:

- **Factorized Covariance Quantization**: Direct quantization of the full 4D covariances of the Gaussians results in severe quality degradation due to their numerical illnesses in long dynamic scenes. We factorize the covariance into four components and quantize them separately, enabling both high compression and controllable approximation error.
- **Layered Compression**: Different temporal layers exhibit diverse temporal extents (ranging from milliseconds to minutes) and statistical properties. We thus perform compression of Gaussian attributes in a layer-wise manner, yielding significantly better quality than global quantization approaches.
- **Residual Codebook Quantization**: To support a large number of Gaussians within a layer without expanding the VQ codebook size, we introduce residual codebook quantization. This allows for fine-grained refinement of quantized components while maintaining a compact representation.

The whole framework is developed in C++/CUDA, accelerating the training, rendering, and compressing for our 4D Gaussian representation. We evaluate our approach on two challenging real-world datasets—N3DV[17] and SelfCap[34]—which feature complex human motion and extended temporal sequences. Experimental results demonstrate that our method consistently outperforms existing baselines in both reconstruction fidelity and computational efficiency. Notably, for a 60-second sequence (3600 frames), our system achieves over 20× compression while preserving comparable visual quality, and enables real-time rendering at over 900 FPS on an NVIDIA RTX 5090 GPU.

2. Related Work

In this section, we review relevant methods for novel-view synthesis (NVS), with a particular focus on dynamic scene modeling, and recent progress in dynamic Gaussian splatting compression.

Dynamic NVS Dynamic novel view synthesis (NVS) aims to reconstruct and render photorealistic, time-varying scenes from a sparse set of input views. Recent advances in neural scene representations, particularly Neural Radiance Fields (NeRF)[22], have enabled high-quality rendering of static scenes through learned implicit volumetric functions. Several extensions have adapted NeRFs to dynamic settings, either by implicitly encoding temporal changes using time-conditioned inputs or latent codes[3, 8], or by explicitly learning deformation fields that map time-varying scenes to a canonical space [5, 24–26, 30]. Other approaches decompose scenes into static and dynamic components [28], leverage keyframe-based architectures [1, 17], estimate scene flow [10, 18, 29], or adopt 4D grid-based representations [2, 6, 7, 16, 27, 31]. Despite their diversity,

these methods typically suffer from the slow volume rendering.

Recently, 3D Gaussian Splatting (3DGS) [14] supports real-time performance and rapid convergence, offering a strong foundation for dynamic scene modeling. Building on 3DGS, several recent works have extended the Gaussian framework to dynamic scenes. Gaussian Tracking [21] models temporal evolution by learning per-Gaussian motion and rotation, enabling object-level tracking across time [32]. Other methods [33, 35] incorporate MLP-based deformation fields to predict time-varying positions. Some approaches [4, 36] represent dynamic scenes directly as collections of 4D Gaussians in *XYZT* space, while [34] explores extending this representation to the longer video setting. However, existing methods struggle with long videos due to excessive storage demands, limiting deployment on memory-constrained devices. In contrast, our method maintains high visual fidelity with low storage and fast rendering, making it well-suited for scalable dynamic scene representation.

Dynamic Gaussian Splatting Compression With the growing use of 4D Gaussian Splatting for dynamic scene modeling, recent works have explored compression strategies to improve storage efficiency and scalability. HiFi4G [13] integrates 3D Gaussians with non-rigid tracking and dual-map motion priors, optimized with adaptive regularization for compact representation. TC3DGS [12] compresses temporal redundancy via Gaussian pruning, gradient-aware quantization, and trajectory interpolation. Light4GS [20] adopts spatiotemporal importance pruning and entropy-constrained SH compression guided by a hierarchical context model. QUEEN [9] learns frame-to-frame residuals and applies quantized sparse encoding for online streaming. 4DGS-1K [37] improves rendering speed by removing short-lived Gaussians and caching visibility. 4DGC [11] introduces a rate-aware pipeline that jointly compresses motion grids and residuals through differentiable quantization and motion-aware modeling. These methods tackle dynamic Gaussian compression from different perspectives, but often face trade-offs in generality, latency, or decoding cost. In contrast, our method introduces the effective compression methods, including Factorized Covariance Quantization, Layered Compression, and Residual Codebook Quantization, enabling high compression ratios and real-time rendering on long dynamic scenes.

3. Method

In this section, we begin by reviewing the 4D-Rotor Gaussian Splatting method [4] in Sec.3.1. We then present our layered 4D Gaussian Splatting representation in Sec.3.2, followed by the proposed compression methods in Sec. 3.3.

3.1. Preliminary of 4D-Rotor Gaussian Splatting

4D-Rotor Gaussian Splatting [4] models dynamic scenes as a set of 4D Gaussians. Each 4D Gaussian is defined by a 4D center position $\boldsymbol{\mu}_{4D} = (\mu_x, \mu_y, \mu_z, \mu_t)$ and a 4D covariance matrix $\boldsymbol{\Sigma}_{4D}$:

$$G_{4D}(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_{4D})^T \boldsymbol{\Sigma}_{4D}^{-1}(\mathbf{x}-\boldsymbol{\mu}_{4D})}. \tag{1}$$

The covariance matrix $\boldsymbol{\Sigma}_{4D}$ is further factorized into a 4D scaling matrix $\mathbf{S}_{4D}$ and a 4D rotation matrix $\mathbf{R}_{4D}$:

$$\boldsymbol{\Sigma}_{4D} = \mathbf{R}_{4D}\mathbf{S}_{4D}\mathbf{S}_{4D}^T\mathbf{R}_{4D}^T. \tag{2}$$

Here, $\mathbf{S}_{4D} = \text{diag}(s_x, s_y, s_z, s_t)$ represents anisotropic scaling, and the 4D rotation $\mathbf{R}_{4D}$ is represented by a 4D rotor. A 4D rotor is parameterized by 8 coefficients $(s, b_{01}, b_{02}, b_{12}, b_{03}, b_{13}, b_{23}, p)$, where the first 4 coefficients encode spatial rotation and the last 4 encode spatiotemporal rotation.

Given a time t, slicing the 4D Gaussian yields a 3D Gaussian in the 3D space:

$$G_{3D}(\mathbf{x}, t) = \mathrm{e}^{-\frac{1}{2}\lambda(t-\mu_t)^2}\mathrm{e}^{-\frac{1}{2}[\mathbf{x}-\boldsymbol{\mu}(t)]^T \boldsymbol{\Sigma}_{3D}^{-1}[\mathbf{x}-\boldsymbol{\mu}(t)]}. \tag{3}$$

Gaussians far from t are culled using a visibility gate: they drop terms with $\lambda(t-\mu_t)^2 > 16$. Equivalently, the effective temporal extent is $\tau = 2\sqrt{16/\lambda}$, which specifies the time span over which a Gaussian contributes effectively.

3.2. Layered Representation

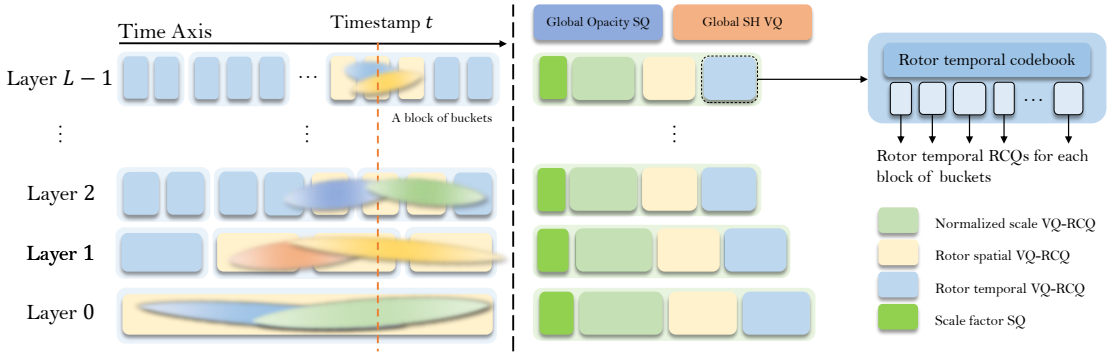
3.2.1. Layer and Bucket Structure

Temporal Gaussian Hierarchy (TGH) [34] organizes 4D Gaussians into multiple temporal levels based on each primitive's temporal extent τ, allocating primitives with long time spans to slowly varying regions and rendering only the relevant segments per timestamp. This models temporal redundancy and reduces both model size and runtime memory.

Although the hierarchical representation in TGH alleviates the challenges of training on long videos, it restricts 4D Gaussians to fixed time segments. Short-τ Gaussians that cross segment boundaries are pushed to lower layers and mixed with geometrically different content, which makes compression harder.

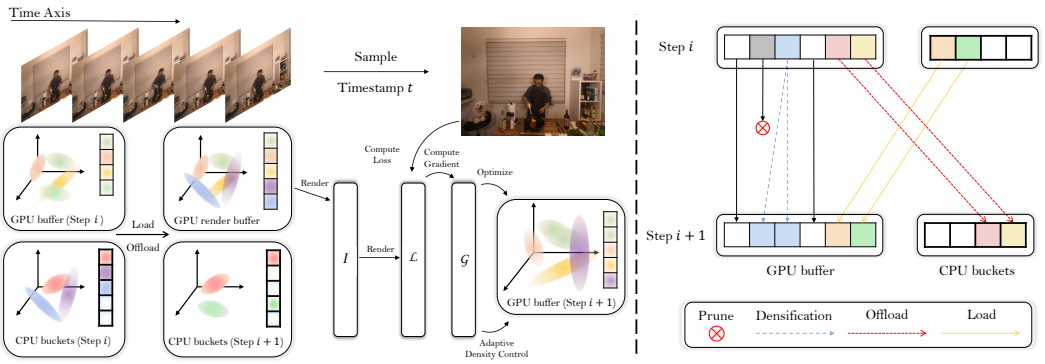
To address this limitation, we propose a layer–bucket structure that splits each TGH segment into left/right buckets while allowing Gaussians to span bucket boundaries. Each Gaussian is assigned using its mean time t and temporal scale τ, avoiding artificial clipping. As shown in Fig. 2, during rendering at a query timestamp, we load only the current bucket and its immediate neighbors in each layer, guaranteeing visibility while minimizing memory.

Rather than manually fixing the number of layers as in TGH, we set $L = \lceil \log_2 n \rceil + 1$, where n is the number of frames in the sequence.



(a) Layered Bucket Structure (b) Hierarchy Compression Structure

Figure 2. **Layered Representation Overview.** As shown in (a), we seperate 4D Gaussians into layers and buckets according to their temporal extents and μ_t. For timestamp t, only the buckets and their neighbors in each layer are required in rendering. (b) shows our compression structure, where appearance attributes are compressed with global opacity SQ and SH VQ, and factorized covariance quantization are then applied for each layer, enhanced with RCQ for blocks of buckets.



(a) Training Framework (b) GPU Buffer Update

Figure 3. **Training Framework Overview.** We implemented a highly optimized training framework (a) adapted for our layered representation. (b) shows that Gaussians' loading and offloading is combined with adaptive density control, minimizing the requirements of frequent memory copy between CPU buckets and GPU buffers.

3.2.2. Training Framework

To improve training efficiency, we propose a Triple-buffer strategy, containing a GPU double buffer and a CPU bucket buffer. As shown in Fig. 3, each training iteration proceeds as follows:

1. Sample a training image with timestamp t.
2. Update GPU buffers:
   - Apply per-step adaptive density control (pruning and densification).
   - Load Gaussians visible at time t from the CPU bucket buffer into the GPU render buffer.
   - Offload Gaussians that have been unused for several steps from GPU to CPU buckets.
3. Run the forward and backward passes, then take an optimizer step.

By maintaining a GPU double buffer, our training framework significantly reduce the cost of CPU-to-GPU memory transfers, which is a major bottleneck in TGH.

Moreover, in vanilla 3DGS, rotation and scale are purely spatial, so a single learning rate for rotation is sufficient. In 4D-Rotor Gaussians, however, a large learning rate on the temporal component of rotor in a static 4D Gaussian causes instability during optimization steps and large position drift at times far from its mean. This is minor on short clips but pronounced on long videos. We therefore adopt a **Dynamic-Aware Rotor Learning Rate (DARLR)**: assign smaller temporal-rotor learning rates to Gaussians with larger temporal extent τ, stabilizing static scene regions during training.

3.3. Compression

3.3.1. Factorized Covariance Quantization

C3DGS [23] directly applies vector quantization (VQ) to normalized 3D covariances. However, we observe that directly applying VQ to normalized 4D covariances fails. This is because the spatial and temporal scales of 4D Gaussians span several orders of magnitude (from 10^{-3} to 10^{3}), and since covariance is quadratically related to the scale (refer to Eq. 2), the effective value range is amplified to more than 10 orders of magnitude. This range far exceeds the representational precision of VQ, leading to infeasible direct quantization.

To tackle this, we propose a factorization strategy tailored to the characteristics of the 4D scaling and rotor.

1. **Scale Decomposition:** We first normalize the 4D scale to obtain a *scale factor* and a *normalized scale*. The scale factor is quantized using scalar quantization (SQ), while the normalized scale is quantized using vector quantization (VQ).
2. **Rotor Decomposition:** For static scene regions, the spatial component of the rotor is isomorphic to a quaternion, and the temporal component is typically close to zero. Therefore, we apply VQ independently to the spatial and temporal components. During decoding, two indices are used to look up the spatial and temporal components from separate codebooks, and the resulting components are merged and normalized to recover the full rotor.

Following C3DGS, we determine VQ importance weights by rendering all training images over time, using the pixel contribution (obtained via backpropagation gradients) of each Gaussian as its quantization weight.

For the scale factor, we apply SQ to the pre-activation value to avoid precision loss due to a large dynamic range, which could exceed the effective granularity of 8-bit scalar quantization.

3.3.2. Layered Compression

Compared to short videos, long videos pose a unique challenge in that the temporal extents of different hierarchy layers span multiple orders of magnitude. This results in significant distributional differences in the scale factor, normalized scale, rotor spatial, and rotor temporal components across layers. If these components are quantized jointly using a shared codebook, the resulting VQs suffer from large codebook sizes and increased quantization errors due to the wide distributional variance.

To address this, we compress these data in a layer-wise manner for those components whose distributions vary significantly across layers. This strategy enables more efficient and accurate compression within each layer's distributional range. In contrast, components such as spherical harmonics (SH) coefficients and opacity, which exhibit relatively consistent distributions across layers, are compressed using global VQ and SQ, respectively. This substantially reduces the storage requirements, particularly for SH codebooks. For opacity, we apply SQ to the sigmoid-activated values, as they naturally lie within the bounded range of $[0, 1]$, making them well-suited for scalar quantization.

Furthermore, to reduce redundancy in both the number of layers and codebooks, we merge the final layers that contain relatively few 4D Gaussians. This merging strategy helps to further compress the representation without compromising the expressiveness of the model.

3.3.3. Residual Codebook Quantization

When modeling extremely long scenes, even layer-wise compression may encounter performance bottlenecks due to limited capacity. To further improve the upper bound of compression performance at minimal cost, we propose Residual Codebook Quantization (RCQ). Specifically, after performing global VQ within each layer, we divide the Gaussians in each layer's bucket into multiple *bucket blocks*. For each bucket block, we perform an additional VQ on the Gaussians within that block to obtain a *block-specific codebook*.

To avoid the additional storage overhead of maintaining a separate codebook for each block, RCQ introduces a lightweight residual codebook to quantize the difference between the block-specific codebook and the original layer-global codebook. This process yields an index table referencing both the global and residual codebooks.

During decoding, we reconstruct each block-specific codebook entry by retrieving a vector from the global codebook and adding its corresponding residual vector from the residual codebook, as specified by the index table. Gaussians within the block can then index into this reconstructed block codebook for rendering.

Finally, we note that layer 0 is excluded from RCQ, as it typically contains fewer temporal variations and benefits less from this additional compression step.

3.4. Implementation Details

We implement the whole pipeline, including training, compression, and rendering, in C++/CUDA for high efficiency. On N3DV with a single NVIDIA RTX 3090, training converges in approximately 30 minutes with a peak GPU memory of only 2 GB; compression takes about 4 minutes. At inference, the renderer sustains 660 FPS, enabling both offline processing and interactive use.

4. Experiments

4.1. Datasets

We evaluate our method on two real-world dynamic scene datasets: N3DV [17] and SelfCap [34]. **N3DV** contains 6 dynamic scenes, each captured by 19–21 cameras at a resolution of 1352×1014 with 10s duration. **SelfCap** includes 6

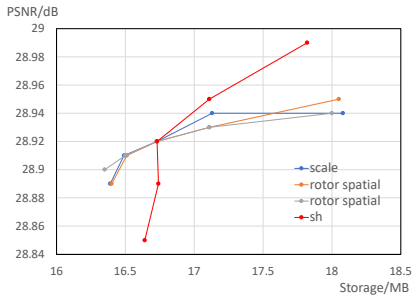


Figure 4. **Ablation on VQ codebook size.** We vary each VQ's codebook from 1,024 to 16,384. Storage grows with size, while the compression visual quality benefits the most for SH VQ compared to other VQs.

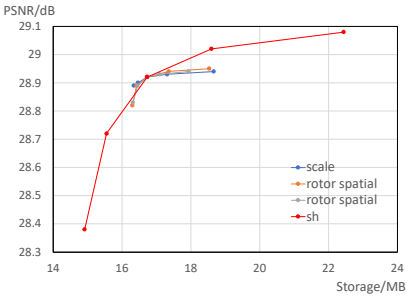


Figure 5. **Ablation on VQ threshold.** We set different thresholds for VQs. Decreasing the threshold increases storage but improves fidelity; consistent with the VQ codebook size study, the compression visual quality benefits most for SH VQ compared to others.

Table 1. **Evaluation on N3DV Dataset**. We compare our method with both NeRF-based and Gaussian-based approaches on an NVIDIA RTX 3090. *: Compression time included. †: Evaluated on the *Flame Salmon* scene only. ‡: Evaluated using a different LPIPS protocol in the original paper.

ID	Method	PSNR↑	SSIM↑	LPIPS↓	Train↓	FPS↑	Storage
a	DyNeRF [17]	29.58	-	0.08	1344 h	0.015	28 MB
b	StreamRF [16]	28.16	0.85	0.31	1.3 h	8.50	24 MB
c	HyperReel [1]	30.36	0.92	0.17	9 h	2.00	360 MB
d	NeRFPlayer [28]	30.69	-	0.11	6 h	0.05	-
e	K-Planes [6]	30.73	0.93	0.07	3.2 h	0.10	309 MB
f	MixVoxels [31]	30.85	0.96	0.21	1.5 h	16.70	508 MB
g	Deformable4DGS [33]	28.42	0.92	0.17	1.2 h	39.93	147 MB
h	RealTime4DGS [36]	31.57	0.97	0.16	8 h	72.80	3128 MB
i	MEGA [38]	31.49	0.97	0.057‡	-	77.42	25.05 MB
j	STG [19]	32.05	0.95	0.14	1.3 h	140	200 MB
k	Ex4DGS [15]	32.11	0.94	0.14	0.6 h	120.6	115 MB
l	TGH† [34]	29.44	0.945	0.214	2.1 h	550	90 MB
m	Ours	32.23	0.941	0.153	0.5 h	351.12	180.7 MB
n	Ours Large	32.06	0.939	0.153	0.6 h*	661.93	13.8 MB
o	Ours Small	31.84	0.937	0.156	0.6 h*	660.79	8.8 MB

scenes featuring large motions such as bicycle repairing and yoga. The sequences are captured at 60 FPS in 4K resolution using 18–22 cameras with long durations (1-10min).

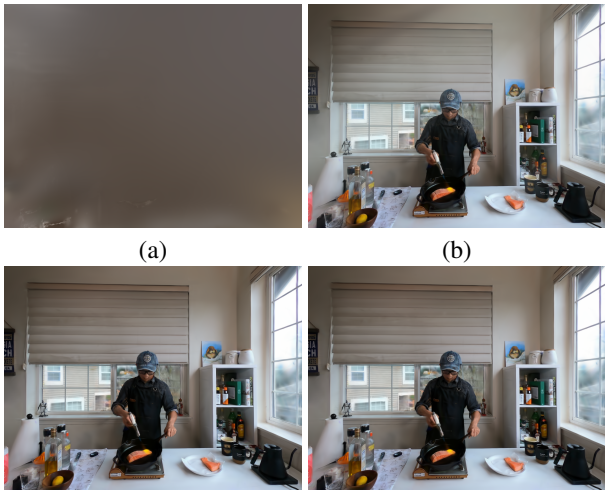


Figure 6. **Ablation of Compression Components.** (a) Direct VQ on 4D covariance (Cov4D) fails to compress geometry attributes and degrades quality; (b) FCQ enables effective geometry compression; (c) adding the layered structure further improves fidelity; (d) RCQ provides an additional gain.

Figure 7. **Ablation of Dynamic-Aware Rotor Learning Rate (DARLR).** DARLR preserves fine detail in static regions; without it, textures appear over-smoothed and lose high-frequency structure.

4.2. Experimental Results

4.2.1. Evaluation on N3DV Dataset

As summarized in Table 1, our method consistently outperforms previous NeRF- and Gaussian-based approaches on the N3DV dataset. The uncompressed variant **Ours** (*l*) achieves the best visual quality but requires 180.7 MB of storage. The compressed variants offer more favorable quality–efficiency trade-offs. **Ours Large** (*m*) uses a 4-minute compression stage and achieves a 13.1× reduction in storage and about 90% faster rendering, with only a 0.17 dB decrease in PSNR. **Ours Small** (*n*) further increases the compression ratio to 20.5× and reduces the bitrate (i.e., Storage/Duration) to below 1 MB/s, while maintaining comparable visual quality to strong baselines. Further compression evaluations are detailed in the supplement.

Overall, our method preserves high visual fidelity and real-time rendering speed under aggressive compression,

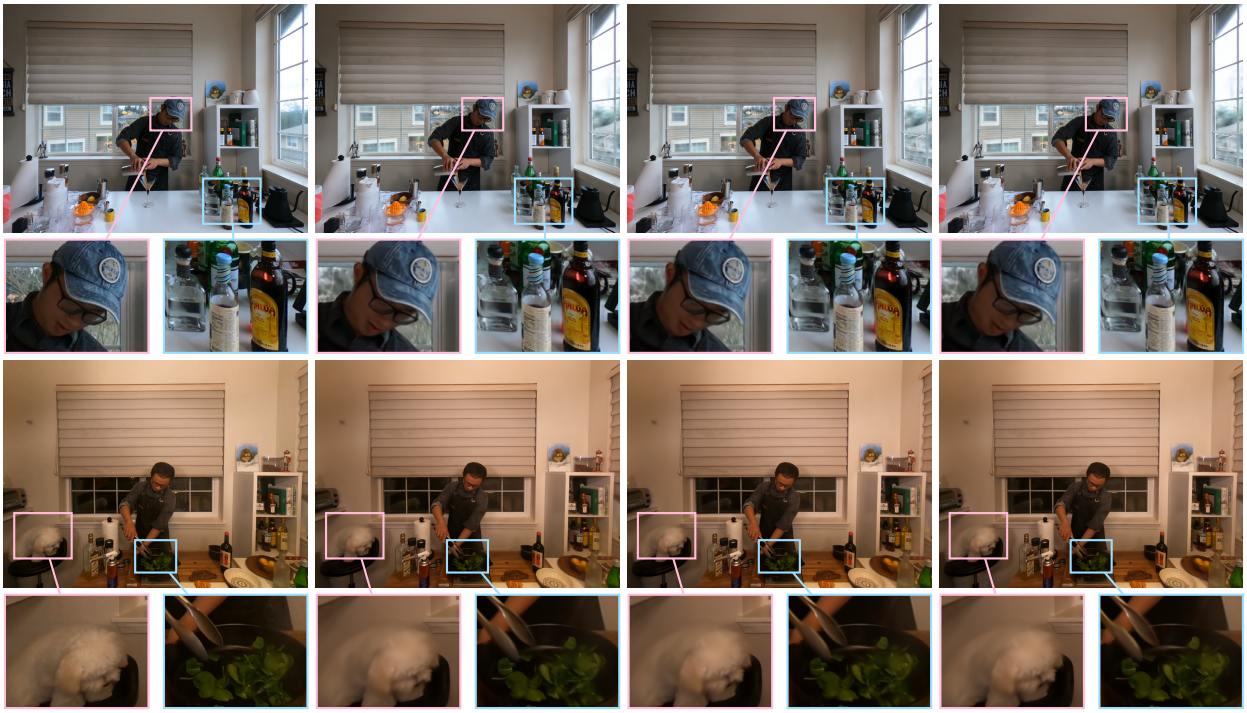


Figure 8. **Qualitative Comparison on N3DV dataset.** Both compressed variants, **Ours Large** and **Ours Small**, closely match the uncompressed model, indicating high visual fidelity under compression.

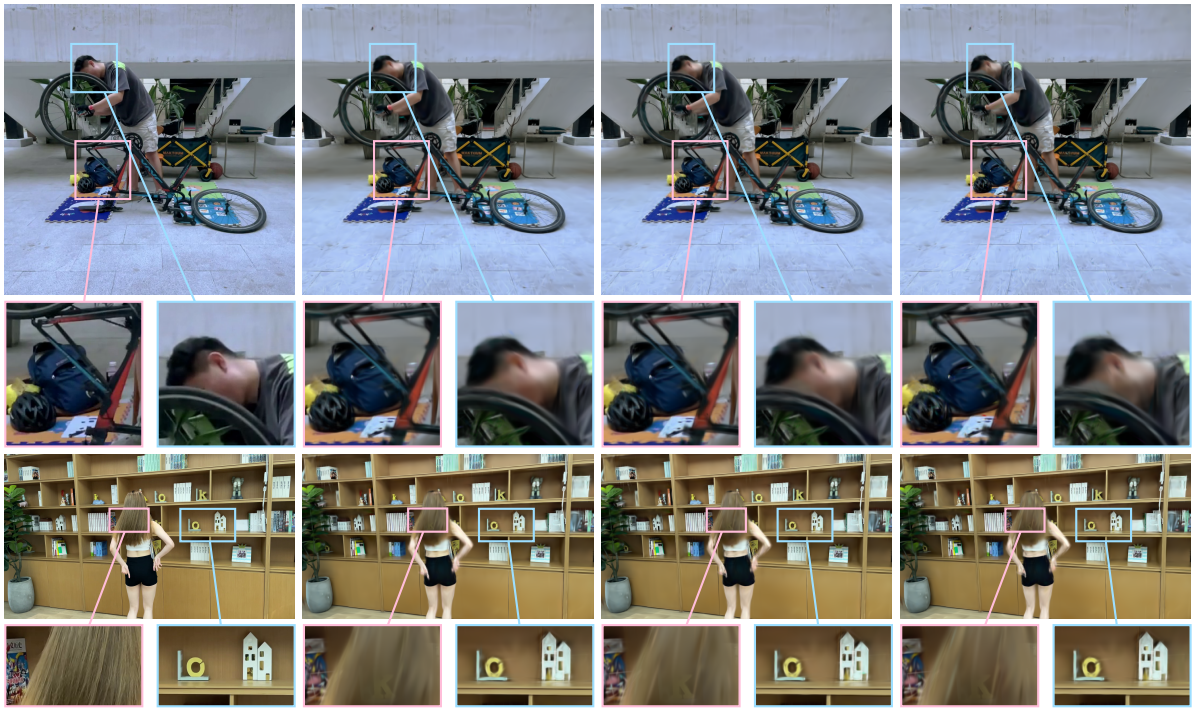


Figure 9. **Qualitative Comparison on SelfCap dataset.** For these long dynamic sequences, **Ours Large** is visually indistinguishable from the uncompressed model, while **Ours Small** shows minor artifacts in highly dynamic regions due to the tighter storage budget.

Table 2. **Quantitative results on the full SelfCap dataset.** We evaluate our method and its compressed variants across all 6 scenes. All metrics are computed over the complete sequences and evaluated on an NVIDIA RTX 5090 GPU.

Method	PSNR ↑	FPS↑	Train ↓	Storage ↓
Ours	24.64	854.21	7.2 h	928.8 MB
Ours Large	24.49	1190.15	7.9 h	48.4 MB
Ours Small	24.41	1193.50	7.9 h	41.8 MB

Table 3. **Quantitative Results on diffenent video duration.** Quantitative results across varying sequence lengths show that our training remains stable as duration increases.

Method/Duration	1 min	2.5 min	5 min	10 min
Ours(PSNR)	26.45	26.29	28.58	24.90
Ours Large(PSNR)	26.26	26.16	28.25	24.73
Ours Small(PSNR)	26.17	26.05	27.81	24.70
Ours(Storage)	578.4 MB	1519.0MB	1151.1 MB	1787.1 MB
Ours Large(Storage)	29.5 MB	55.6 MB	56.3 MB	92.9 MB
Ours Small(Storage)	27.2 MB	49.0 MB	51.8 MB	87.4 MB

Table 4. **Ablation of compression components.** Quality improves progressively as components are added, with the largest gains from FCQ and the layered structure. The RCQ codebook size is set to 256.

	Method	PSNR	Storage
(a)	Cov4D VQ	11.35	29.03 MB
(b)	+ FCQ	22.09	16.22 MB
(c)	+ Layer Structure	28.90	15.93 MB
(d)	+ RCQ	28.92	16.73 MB

making it suitable for practical deployment.

4.2.2. Evaluation on SelfCap Dataset

Most prior dynamic NVS baselines handle only around 10s clips. To assess long-video effectiveness, we evaluate all six SelfCap scenes; due to the lack of open-source long-video baselines, we compare our uncompressed and compressed variants only. Comparisons against short-sequence baselines on 1s clips are in the supplement.

As shown in Table 2, compression preserves visual quality while markedly reducing storage and improving rendering, yielding a 40% FPS increase. **Ours Large** attains higher fidelity, whereas **Ours Small** pushes the compression ratio to $19.1\times$ with competitive quality. Fig. 9 further illustrates that our method maintains high fidelity in regions with rapid human–object motion, with compressed outputs closely matching the uncompressed version. These results validate our representation's suitability for long dynamic scenes, delivering practical storage and speed gains without compromising visual quality.

To further assess long-sequence reconstruction, we use the SelfCap *Bike* scene. As shown in Table 3, we form 60-frame clips with increasing temporal span for training and evaluation. Results show consistent performance across durations, confirming robustness to long videos.

4.3. Ablation Studies

We conduct ablation studies to evaluate the effectiveness of individual components in our method on N3DV *Flame Salmon* and SelfCap *Bike*.

Table 5. **Ablation on RCQ codebook size.** We validate our RCQ method with different codebook sizes. Results show that RCQ keeps stable under different codebook sizes.

RCQ codebook size	PSNR	Storage
64	28.919	16.740 MB
128	28.919	16.734 MB
256	28.922	16.733 MB
512	28.923	16.734 MB
1024	28.921	16.733 MB

Components in Compression We ablate our compression pipeline on N3DV *Flame Salmon* (Table 4, Fig. 6). Naïvely compressing the 4D covariance severely degrades fidelity. Adding FCQ lifts PSNR from 11.35 to 22.09 while shrinking storage from 29.03 MB to 16.22 MB. Layered compression further raises PSNR to 28.90 by preserving fine-grained structure. Finally, RCQ brings an additional quality gain with a small storage increase.

Hyperparameters in Compression We analyze the impact of key hyperparameters in compression on N3DV *Flame Salmon*. As shown in Fig. 4, increasing the **VQ codebook size** from 1,024 to 16,384 raises storage but consistently improves PSNR by reducing quantization error; among Gaussian attributes, enlarging the **SH codebook size** yields the highest quality-per-storage gain (PSNR per MB). Fig. 5 shows a similar trend for the **VQ threshold**: lowering the threshold for SH features offers the best quality–storage trade-off. Finally, Table 5 shows that varying the RCQ codebook size has only minor effects on storage and fidelity, indicating robustness across settings.

Dynamic-Aware Rotor Learning Rate We introduce a Dynamic-Aware Rotor Learning Rate (DARLR) strategy during training. As shown in Fig. 7, DARLR preserves fine textures and significantly reduces artifacts in static regions, yielding higher visual fidelity.

5. Conclusions

We propose L4DRotorGS, a compact and efficient representation for long dynamic scenes based on layered 4D-Rotor Gaussians. Our method significantly reduces GPU memory usage and storage cost while maintaining high visual quality and real-time rendering performance. Experiments demonstrate consistent improvements over prior methods in memory efficiency, rendering quality, and scalability.

While our method achieves high-quality visual results, real-time rendering performance, and low VRAM consumption, it still has several limitations. The compression process is still time-consuming, and our current framework does not support online training, which limits its adaptability in streaming capture scenarios.

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*, pages 16610–16620, 2023. 2, 6

[2] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 130–141, 2023. 2

[3] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14304–14314. IEEE Computer Society, 2021. 2

[4] Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 4d-rotor gaussian splatting: towards efficient novel view synthesis for dynamic scenes. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024. 2, 3

[5] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022. 2

[6] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12479–12488, 2023. 2, 6

[7] Wanshui Gan, Hongbin Xu, Yi Huang, Shifeng Chen, and Naoto Yokoya. V4d: Voxel for 4d novel view synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 2023. 2

[8] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021. 2

[9] Sharath Girish, Tianye Li, Amrita Mazumdar, Abhinav Shrivastava, Shalini De Mello, et al. Queen: Quantized efficient encoding of dynamic gaussians for streaming free-viewpoint videos. *Advances in Neural Information Processing Systems*, 37:43435–43467, 2024. 3

[10] Xiang Guo, Jiadai Sun, Yuchao Dai, Guanying Chen, Xiaoqing Ye, Xiao Tan, Errui Ding, Yumeng Zhang, and Jingdong Wang. Forward flow for novel view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16022–16033, 2023. 2

[11] Qiang Hu, Zihan Zheng, Houqiang Zhong, Sihua Fu, Li Song, Guangtao Zhai, Yanfeng Wang, et al. 4dgc: Rate-aware 4d gaussian compression for efficient streamable free-viewpoint video. *arXiv preprint arXiv:2503.18421*, 2025. 3

[12] Saqib Javed, Ahmad Jarrar Khan, Corentin Dumery, Chen Zhao, and Mathieu Salzmann. Temporally compressed 3d gaussian splatting for dynamic scenes. *arXiv preprint arXiv:2412.05700*, 2024. 3

[13] Yuheng Jiang, Zhehao Shen, Penghao Wang, Zhuo Su, Yu Hong, Yingliang Zhang, Jingyi Yu, and Lan Xu. Hifi4g: High-fidelity human performance rendering via compact gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19734–19745, 2024. 3

[14] 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):1–14, 2023. 3

[15] Junoh Lee, ChangYeon Won, Hyunjun Jung, Inhwan Bae, and Hae-Gon Jeon. Fully explicit dynamic gaussian splatting. *Advances in Neural Information Processing Systems*, 37:5384–5409, 2024. 6

[16] Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Ping Tan. Streaming radiance fields for 3d video synthesis. *Advances in Neural Information Processing Systems*, 35: 13485–13498, 2022. 2, 6

[17] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5521–5531, 2022. 1, 2, 5, 6

[18] 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*, pages 6498–6508, 2021. 2

[19] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. Spacetime gaussian feature splatting for real-time dynamic view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8508–8520, 2024. 6

[20] Mufan Liu, Qi Yang, He Huang, Wenjie Huang, Zhenlong Yuan, Zhu Li, and Yiling Xu. Light4gs: Lightweight compact 4d gaussian splatting generation via context model. *arXiv preprint arXiv:2503.13948*, 2025. 3

[21] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. *arXiv preprint arXiv:2308.09713*, 2023. 3

[22] B Mildenhall, PP Srinivasan, M Tancik, JT Barron, R Ramamoorthi, and R Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, 2020. 2

[23] 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*, pages 10349–10358, 2024. 5

[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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021. 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 Transactions on Graphics (TOG)*, 40 (6):1–12, 2021.

[26] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021. 2

[27] Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. Tensor4d: Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16632–16642, 2023. 2

[28] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. Nerfplayer: A streamable dynamic scene representation with decomposed neural radiance fields. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2732–2742, 2023. 2, 6

[29] Fengrui Tian, Shaoyi Du, and Yueqi Duan. Mononerf: Learning a generalizable dynamic radiance field from monocular videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17903–17913, 2023. 2

[30] Edith Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12959–12970, 2021. 2

[31] Feng Wang, Sinan Tan, Xinghang Li, Zeyue Tian, Yafei Song, and Huaping Liu. Mixed neural voxels for fast multi-view video synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19706–19716, 2023. 2, 6

[32] Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. Tracking everything everywhere all at once. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19795–19806, 2023. 3

[33] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. *arXiv preprint arXiv:2310.08528*, 2023. 3, 6

[34] Zhen Xu, Yinghao Xu, Zhiyuan Yu, Sida Peng, Jiaming Sun, Hujun Bao, and Xiaowei Zhou. Representing long volumetric video with temporal gaussian hierarchy. *ACM Transactions on Graphics (TOG)*, 43(6):1–18, 2024. 2, 3, 5, 6

[35] 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. 3

[36] Zeyu Yang, Hongye Yang, Zijie Pan, Xiatian Zhu, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *International Conference on Learning Representations (ICLR)*, 2024. 2, 3, 6

[37] Yuheng Yuan, Qiuhong Shen, Xingyi Yang, and Xinchao Wang. 1000+ fps 4d gaussian splatting for dynamic scene rendering. *arXiv preprint arXiv:2503.16422*, 2025. 3

[38] Xinjie Zhang, Zhening Liu, Yifan Zhang, Xingtong Ge, Dailan He, Tongda Xu, Yan Wang, Zehong Lin, Shuicheng Yan, and Jun Zhang. Mega: Memory-efficient 4d gaussian splatting for dynamic scenes. *arXiv preprint arXiv:2410.13613*, 2024. 6

Layered 4D-Rotor Gaussian Splatting: A Compressed Representation for Long Dynamic Scenes — Supplementary Material

Contents

A. Details of RCQ

Starting from layer 1, we perform RCQ for each block of buckets. We carry out global weighted vector quantization for each layer, specifically on the normalized scale, rotor spatial, and rotor temporal components. The codeword $\mathbf{c}_i^g$ assigned by global vector quantization to each Gaussian $\mathbf{g}_i$ in this layer is given by:

$$\mathbf{c}_i^g = \arg\min_{\mathbf{c}\in\mathcal{C}^g} \|\mathbf{g}_i - \mathbf{c}\|_2. \tag{1}$$

Here, $\mathcal{C}^g$ is the global codebook and $\mathbf{c}$ represents each codeword in $\mathcal{C}^g$. Next, we construct a local codebook $\mathcal{C}^l$ for each block of buckets:

$$\mathcal{C}^l = \text{weighted-k-means}\left(\{\mathbf{g}_j \mid \mathbf{g}_j \in B_k\}, K\right), \tag{2}$$

where $\mathbf{g}_j$ denotes a Gaussian in block B_k, and K is the codebook size, which is set to the final size of $\mathcal{C}^g$. Next, we calculate the residual $\mathbf{r}_m$ for each codeword $\mathbf{l}_m$ in $\mathcal{C}^l$ as follows:

$$\forall \mathbf{l}_m \in \mathcal{C}^l, \quad \mathbf{c}_m = \arg\min_{\mathbf{c}\in\mathcal{C}^g} \|\mathbf{l}_m - \mathbf{c}\|_2, \quad \mathbf{r}_m = \mathbf{l}_m - \mathbf{c}_m. \tag{3}$$

Subsequently, we perform residual quantization to obtain the residual codebook $\mathcal{R}^l$:

$$\mathcal{R}^l = \text{weighted-k-means}\left(\{\mathbf{r}_m \mid \mathbf{l}_m \in \mathcal{C}^l\}, M\right), \tag{4}$$

where M is the residual codebook size. The weight w_m for the residual $\mathbf{r}_m$ is defined as the sum of weights of the Gaussians assigned to $\mathbf{l}_m$:

$$w_m = \sum_{\mathbf{g}_j\in\mathcal{S}_m} W_j, \quad \mathcal{S}_m = \left\{\mathbf{g}_j \mid \arg\min_{\mathbf{l}\in\mathcal{C}^l} \|\mathbf{g}_j - \mathbf{l}\|_2 = \mathbf{l}_m\right\}. \tag{5}$$

Here, W_j represents the weight of the Gaussian $\mathbf{g}_j$, and $\mathcal{S}_m$ is the set of Gaussians in the block whose nearest local codeword is $\mathbf{l}_m$.

B. Additional Experiments

B.1. Additional Implementation Details

Initialization. We initialize our model using a colored sparse point cloud generated by COLMAP [2] from the initial frames across all camera views.

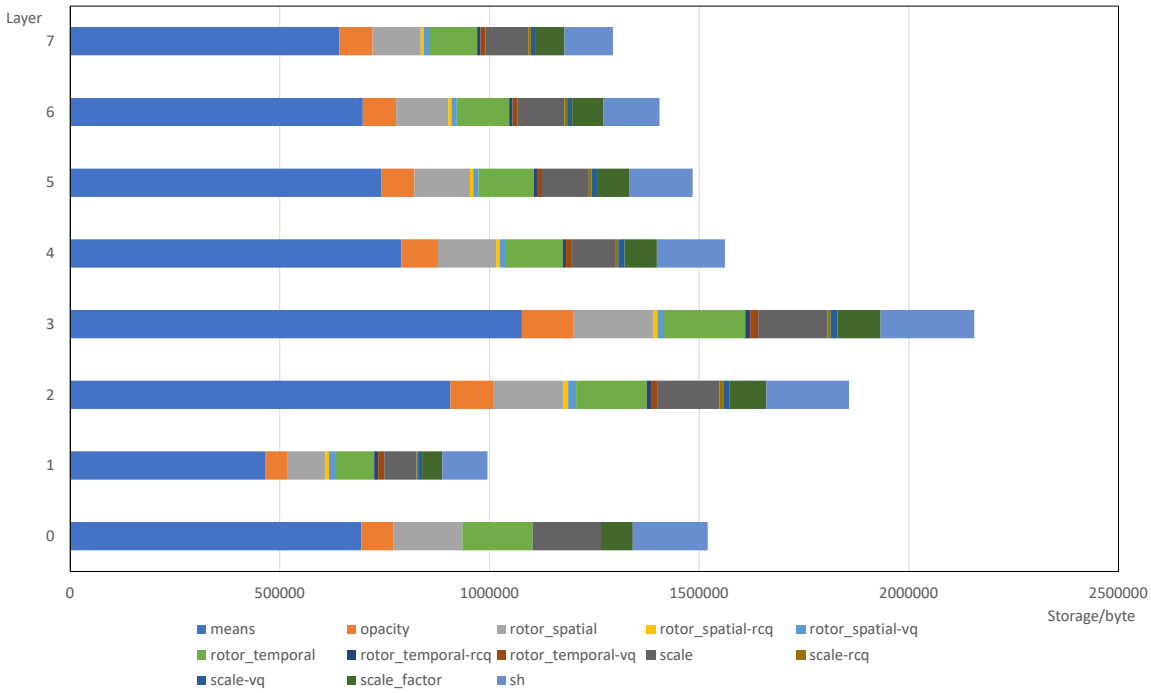


Figure 10. **Compressed Indices Storage per Layer.** We visualize the size of each layer's indices after entropy encoding. The Gaussian means are stored in FP16 precision, which accounts for over 40% of the total index storage.

Training. We optimize our model using Adam for 40K steps on the *10s* scenes, 200K steps on the *40s* scenes from the N3DV dataset, and 2M steps on the *10min Bike* scene from the SelfCap dataset. The settings for our loss functions, learning rates, densification, pruning, and opacity reset follow those of [1, 3].

Compression. Following C3DGS [6], we adopt a compact model configuration for our *Small* setting. Specifically, we use a codebook size of 4096 for all vector-quantized components, including SH, normalized scale, and the spatial and temporal components of the rotor. The quantization thresholds are set to 2×10^{-7} for normalized scale, 1×10^{-6} for the spatial rotor, 8×10^{-6} for the temporal rotor, and 6×10^{-7} for SH. The residual codebook size is fixed at 512 across all layers. In the *Large* model setting, we increase the SH codebook size to 8192 and reduce the SH threshold to 1×10^{-7}, keeping all other settings unchanged.

B.2. Additional Compression Details

B.2.1. Storage Breakdown

Fig. 10 shows the storage consumption of Gaussian indices and means. We store the means using FP16 precision without applying any quantization, as quantizing the means would cause spatial collisions among Gaussian centers, severely degrading the reconstruction accuracy.

Fig. 11 presents the storage breakdown across all codebooks, with SH dominating the storage footprint. This substantial storage requirement aligns with our experimental findings, which demonstrate that SH compression has the most significant impact on rendering quality.

Fig. 12 further illustrates the change in the number of Gaussians across layers before and after compression. Our pruning strategy significantly reduces the total number of Gaussians, directly contributing to the observed FPS improvement. Furthermore, our layer-wise merging strategy effectively reduces the total number of layers by combining those with fewer Gaussians, thereby improving the overall compression ratio.

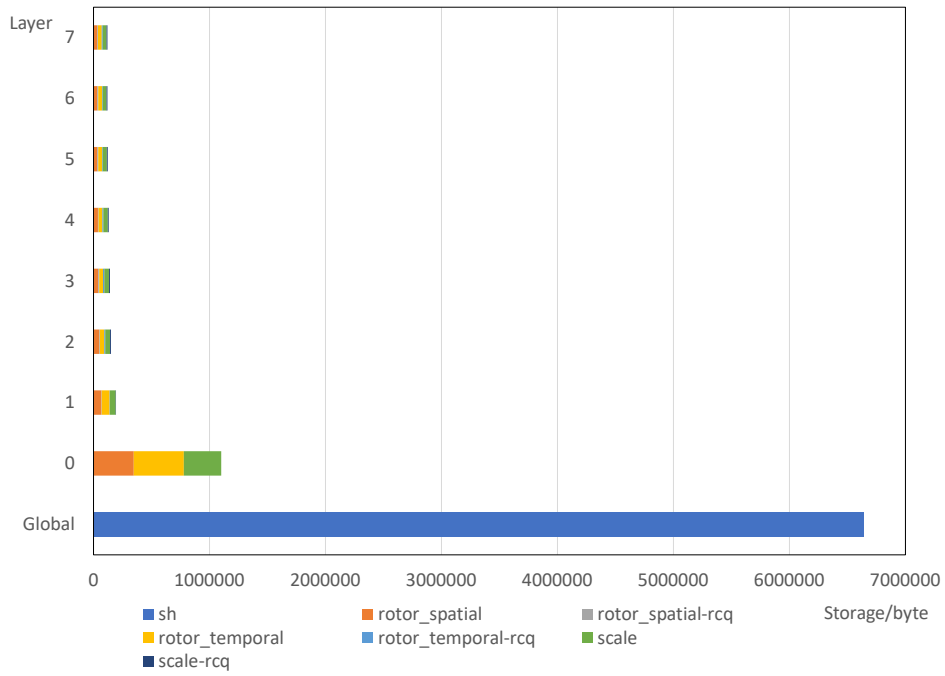


Figure 11. **Codebook Storage per Layer.** We visualize the sizes of the global and layer-wise codebooks. The SH codebooks account for approximately 75% of the total codebook storage, whereas the RCQ codebooks incur minimal storage overhead.

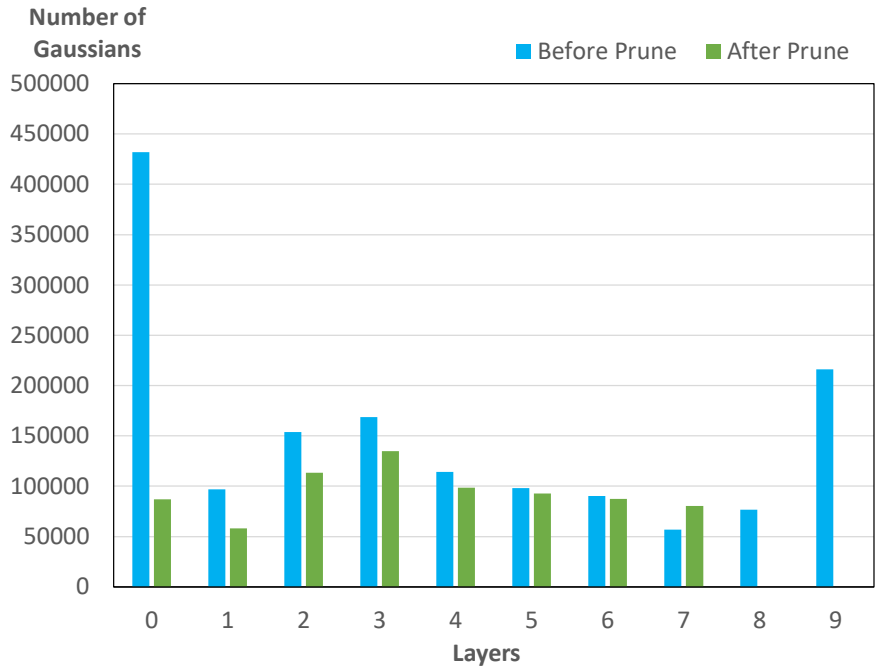


Figure 12. **Number of Gaussians per Layer.** Our compression method prunes Gaussians that are assigned zero weights during vector quantization. This significantly reduces the total number of 4D Gaussians while maintaining rendering quality. Furthermore, the final two layers are merged into layer 7 due to their low Gaussian count after pruning.

B.2.2. Timing Statistics

Tab. 6 details the time cost of our compression pipeline. Notably, the sensitivity calculation stage accounts for 97% of the total processing time.

Table 6. **Time Consumption per Compression Step.** We report the runtime for each step on the 40s *Flame Salmon* scene. Unlike C3DGS [6], our method does not require additional fine-tuning. However, computing sensitivity dominates the time cost due to the massive number of frames (over 20,000). Conversely, the remaining steps are highly efficient thanks to our optimized C++/CUDA implementation. Timings are measured on an NVIDIA RTX 5090 GPU.

Compression Step	Time (s)
Sensitivity Calculation	439.1
Pruning & Merging	0.5
VQ, RCQ, & SQ	12.4

B.2.3. Analysis of Perceptual Fidelity

To assess the temporal stability of our compressed representation, we measure the Just-Objectionable-Difference (JOD) [5] between the rendered videos of our full model ("Ours") and the heavily compressed variant ("Ours Small"). The JOD metric evaluates perceptual video quality on a scale of 0 to 10, where 10 indicates perfect perceptual equivalence. As shown in Table 7, our compressed model achieves high JOD scores on both the *60s Bike* and *40s Flame Salmon* sequences. This confirms that our pipeline effectively preserves temporal coherence despite a significant reduction in storage.

The slight deviation from a perfect JOD score stems primarily from the aggressive quantization of SH coefficients in the "Small" variant. Unlike geometric parameters (scale and rotor) that maintain structural integrity, SH coefficients govern view-dependent color radiance. Compressing them leads to a loss of high-frequency color information, manifesting as minor luminance shifts and reduced contrast. While this degradation is largely imperceptible in static backgrounds, complex motion in dynamic regions perceptually amplifies these color inaccuracies, accounting for the minor temporal inconsistencies.

Table 7. **JOD Evaluation.** We compare the temporal stability between our full model and the heavily compressed variant.

Scene	JOD Score ↑
60s Bike	8.20
40s Flame Salmon	8.86

B.3. Additional Results

B.3.1. Per-Scene Results

Tables 8 and 9 detail the per-scene quantitative results on the N3DV and SelfCap datasets, respectively. For the SelfCap scenes, we adopt the official testing viewpoints where available: `0009` for *Bike*, `0015` for *Hair*, and `0007` for *Corgi*. For the remaining three scenes without officially designated testing views, we explicitly select `09` for *Bar*, `0015` for *Dance*, and `0011` for *Yoga*.

Furthermore, as noted in the original SelfCap dataset, the baseline TGH [9] neither withheld testing views during training nor released quantitative metrics or source code for this dataset. Consequently, we primarily benchmark the compressed variants against our own uncompressed full model.

B.3.2. Comparison on SelfCap Dataset

As noted in the main text, we provide an additional comparison against recent short-sequence baselines on a 1-second clip from the SelfCap dataset [9]. While baselines like FreeTimeGS [8] are heavily optimized for peak fidelity on brief snippets, our representation is inherently designed to scale efficiently across thousands of frames. Despite this architectural difference, Table 10 demonstrates that our method achieves highly competitive visual quality and state-of-the-art rendering speeds, significantly outperforming other approaches in efficiency.

B.3.3. Quantitative Results on HyperNeRF Dataset

To evaluate the capability of our method on casually captured videos, we conduct additional experiments on four scenes (*3D Printer*, *Broom*, *Chicken*, and *Banana*) from the HyperNeRF dataset [7]. As presented in Table 11, our full model achieves

Table 8. **Per-Scene Results on N3DV Dataset.** Quantitative results of our method and its compressed variants across all 6 N3DV scenes. For *Flame Salmon*, we additionally provide a complete 40-second version as a separate comparison.

Method	Ours		Ours Large		Ours Small	
	PSNR	Storage	PSNR	Storage	PSNR	Storage
Flame Steak	34.30	163.9 MB	34.09	13.0 MB	33.86	8.3 MB
Coffee Martini	28.74	206.1 MB	28.60	14.6 MB	28.48	9.5 MB
Sear Steak	34.25	158.3 MB	34.09	12.3 MB	33.79	7.7 MB
Cut Roasted Beef	33.75	170.0 MB	33.55	13.9 MB	33.25	8.5 MB
Cook Spinach	33.39	168.8 MB	33.19	13.8 MB	32.95	8.6 MB
Flame Salmon	28.94	216.9 MB	28.82	14.9 MB	28.72	10.1 MB
*Flame Salmon (40s)**	29.27	372.8 MB	29.07	20.8 MB	28.92	16.7 MB
Average (except *)	32.23	180.7 MB	32.06	13.8 MB	31.84	8.8 MB

Table 9. **Per-Scene Results on SelfCap Dataset.** Quantitative results of our method and its compressed variants across all 6 SelfCap scenes.

Method	Ours		Ours Large		Ours Small	
	PSNR	Storage	PSNR	Storage	PSNR	Storage
Bike (10min)	24.90	1787.1 MB	24.73	92.9 MB	24.70	87.4 MB
Bar	27.56	491.7 MB	27.35	27.2 MB	27.25	23.7 MB
Corgi	23.73	228.1 MB	23.59	16.2 MB	23.56	12.7 MB
Dance	23.91	1380.7 MB	23.70	70.6 MB	23.46	62.3 MB
Yoga	25.00	490.3 MB	24.94	28.8 MB	24.89	22.6 MB
Hair	22.72	1194.7 MB	22.61	54.6 MB	22.60	41.9 MB
Average	24.64	928.8 MB	24.49	48.4 MB	24.41	41.8 MB

Table 10. **Quantitative comparison on the SelfCap Dataset.** All methods are evaluated on a 1-second clip to enable direct comparison with recent short-sequence baselines.

Method	PSNR ↑	FPS ↑
STG [4]	24.97	142
Real-Time 4DGS [10]	25.98	65
FreeTimeGS [8]	**27.41**	467
Ours	26.12	**977**

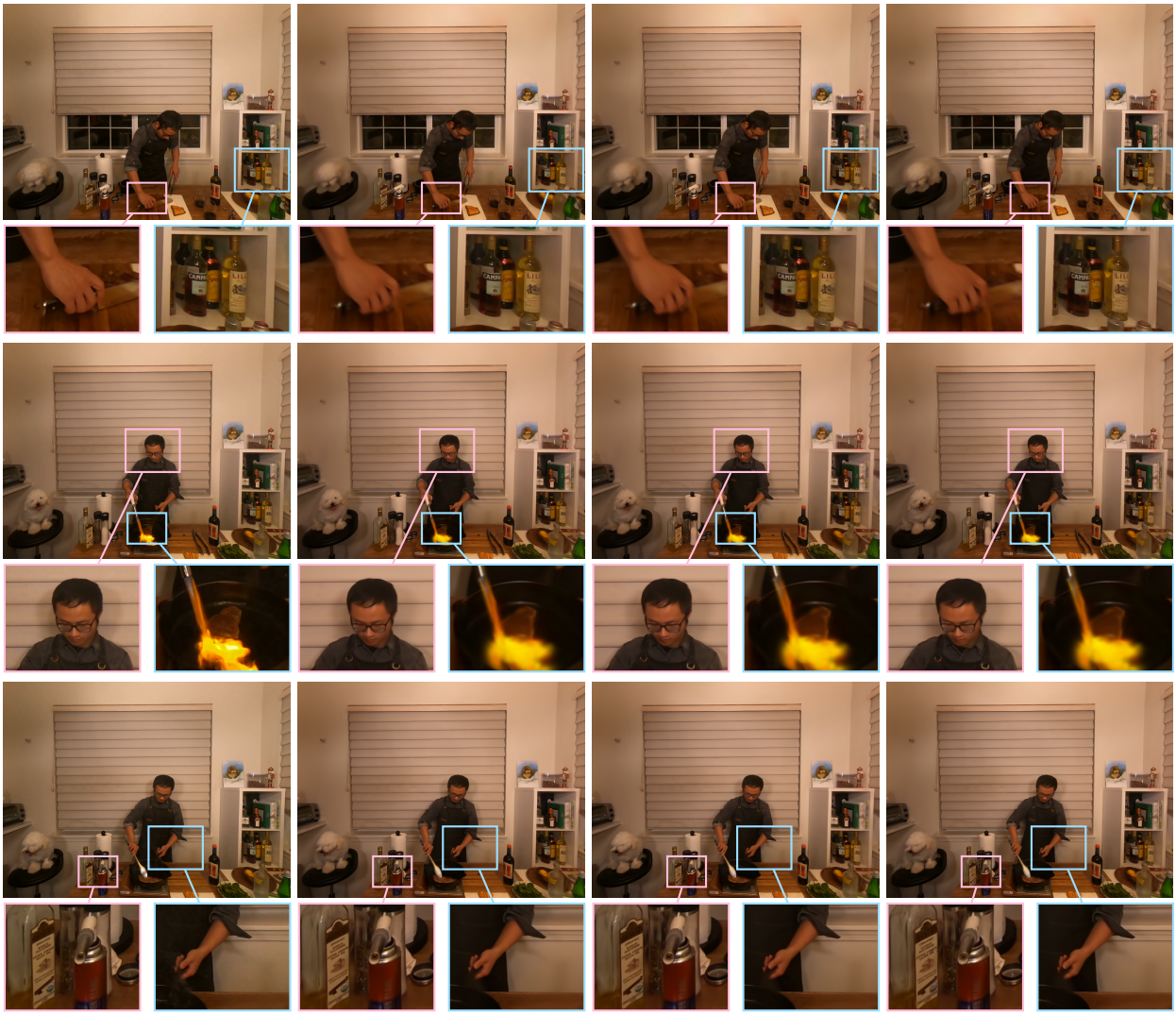
rendering quality comparable to the 4D-rotor GS baseline. More importantly, our compressed variants successfully maintain this competitive visual fidelity while achieving a drastic reduction in storage footprint.

Table 11. **Quantitative results on the HyperNeRF Dataset.** We evaluate our method against the 4D-rotor GS baseline on four casually captured scenes. Our compressed variants achieve massive storage reductions with negligible PSNR drops.

Method	PSNR↑	Storage↓
4D-Rotor GS [1]	25.71	240.8 MB
Ours	**25.84**	219.3 MB
Ours Large	25.69	20.9 MB
Ours Small	25.59	**16.7 MB**

B.3.4. Additional Qualitative Results

We provide further results on the N3DV and SelfCap datasets, illustrated in Fig. 13 and Fig. 14, respectively.



Ground Truth Ours Ours Large Ours Small

Figure 13. **Qualitative Comparison on N3DV Dataset.** Both compressed variants, **Ours Large** and **Ours Small**, closely match the uncompressed model, indicating high visual fidelity under compression.

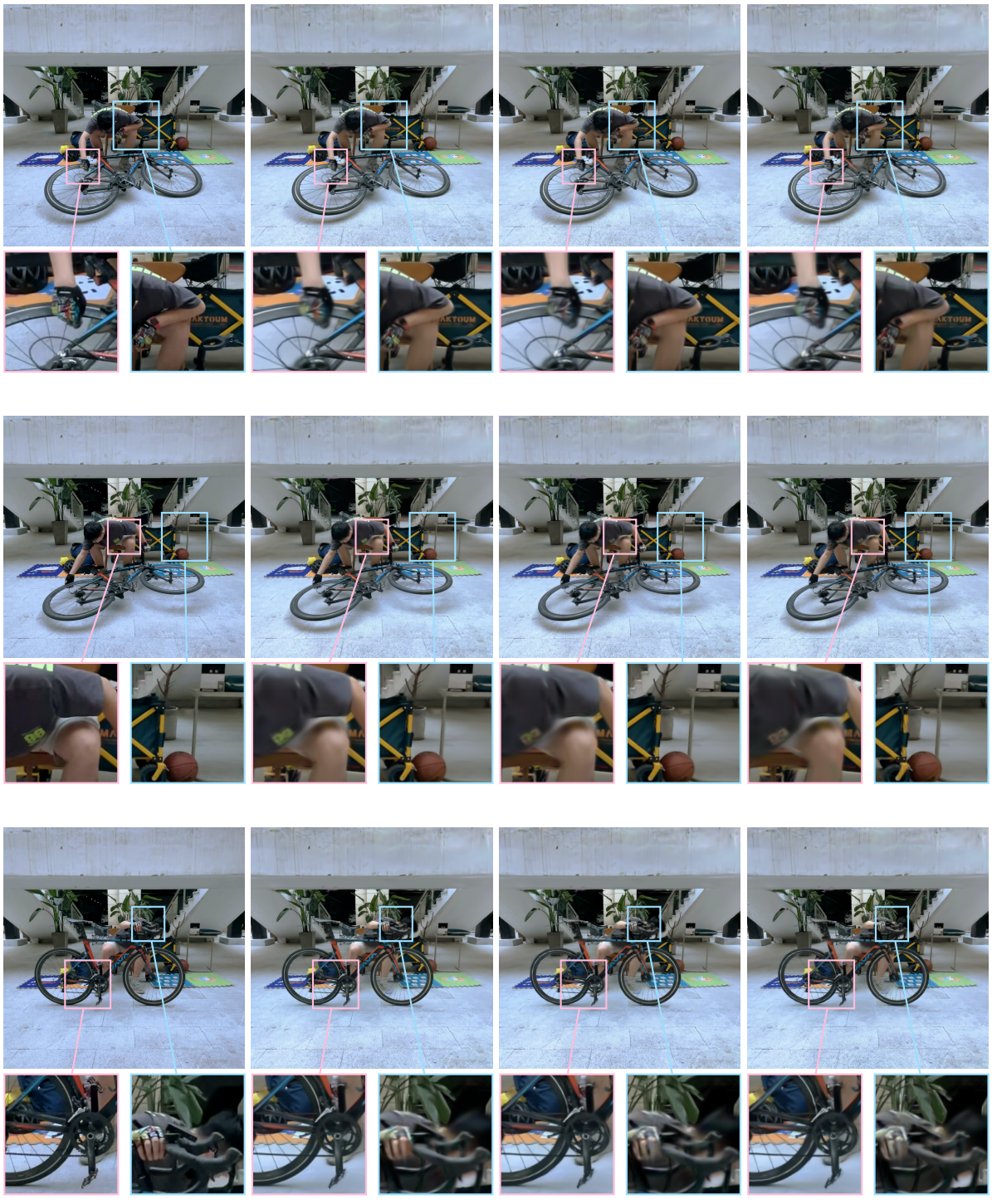


Figure 14. **Qualitative Comparison on SelfCap Dataset.** For these long dynamic sequences, **Ours Large** is visually indistinguishable from the uncompressed model, while **Ours Small** shows minor artifacts in highly dynamic regions due to the tighter storage budget.

References

[1] Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 4d-rotor gaussian splatting: towards efficient novel view synthesis for dynamic scenes. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024. 2, 5

[2] Alex Fisher, Ricardo Cannizzaro, Madeleine Cochrane, Chatura Nagahawatte, and Jennifer L Palmer. Colmap: A memory-efficient occupancy grid mapping framework. *Robotics and Autonomous Systems*, 142:103755, 2021. 1

[3] 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):1–14, 2023. 2

[4] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. Spacetime gaussian feature splatting for real-time dynamic view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8508–8520, 2024. 5

[5] Rafał K Mantiuk, Gyorgy Denes, Alexandre Chapiro, Anton Kaplanyan, Gizem Rufo, Romain Bachy, Trisha Lian, and Anjul Patney. Fovvideovdp: A visible difference predictor for wide field-of-view video. *ACM Transactions on Graphics (TOG)*, 40(4):1–19, 2021. 4

[6] 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*, pages 10349–10358, 2024. 2, 4

[7] 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 Transactions on Graphics (TOG)*, 40(6):1–12, 2021. 4

[8] Yifan Wang, Peishan Yang, Zhen Xu, Jiaming Sun, Zhanhua Zhang, Yong Chen, Hujun Bao, Sida Peng, and Xiaowei Zhou. Freetimegs: Free gaussian primitives at anytime anywhere for dynamic scene reconstruction. In *CVPR*, 2025. 4, 5

[9] Zhen Xu, Yinghao Xu, Zhiyuan Yu, Sida Peng, Jiaming Sun, Hujun Bao, and Xiaowei Zhou. Representing long volumetric video with temporal gaussian hierarchy. *ACM Transactions on Graphics (TOG)*, 43(6):1–18, 2024. 4

[10] Zeyu Yang, Hongye Yang, Zijie Pan, Xiatian Zhu, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *International Conference on Learning Representations (ICLR)*, 2024. 5