4D Gaussian Videos with Motion Layering

PINXUAN DAI[*], State Key Lab of CAD&CG, Zhejiang University, China
PEIQUAN ZHANG[*], State Key Lab of CAD&CG, Zhejiang University, China
ZHENG DONG, State Key Lab of CAD&CG, Zhejiang University, China
KE XU, City University of Hong Kong, China
YIFAN PENG, The University of Hong Kong, China
DANDAN DING, Hangzhou Normal University, China
YUJUN SHEN, Ant Group, China
YIN YANG, The University of Utah, USA
XINGUO LIU, State Key Lab of CAD&CG, Zhejiang University, China
RYNSON W.H. LAU, City University of Hong Kong, China
WEIWEI XU[†], State Key Lab of CAD&CG, Zhejiang University, China

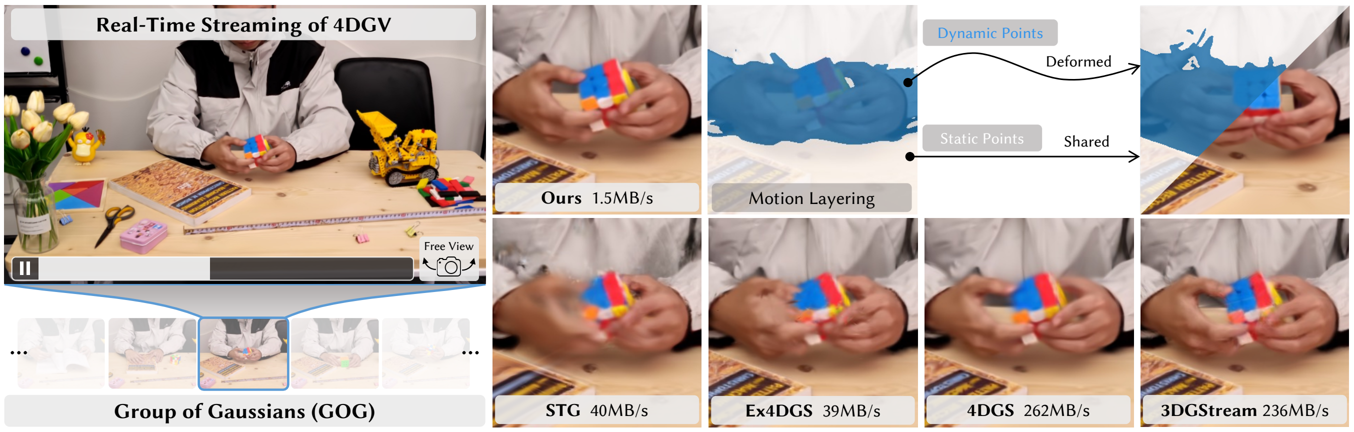


Fig. 1. **4D Gaussian Video (4DGV) for volumetric video streaming.** Left: Our 4DGV representation reconstructs high-quality volumetric video content using Group of Gaussians (GOG). Notably, the reconstructed GOGs occupy a compact size of only 1~4MB per second across diverse datasets, which can be displayed in our web player in real-time using Github as the HTTP server.

[*]Joint first authors
[†]Corresponding author

Authors' addresses: Pinxuan Dai, daipinxuan@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Peiquan Zhang, pqcheung@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Zheng Dong, zhengdong@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Ke Xu, kkangwing@gmail.com, City University of Hong Kong, China; Yifan Peng, evanpeng@hku.hk, The University of Hong Kong, China; Dandan Ding, dandanding@hznu.edu.cn, Hangzhou Normal University, China; Yujun Shen, shenyujun0302@gmail.com, Ant Group, China; Yin Yang, yangzzzy@gmail.com, The University of Utah, USA; Xinguo Liu, xgliu@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Rynson W.H. Lau, rynson.lau@cityu.edu.hk, City University of Hong Kong, China; Weiwei Xu, xww@cad.zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China.

ACM 0730-0301/2025/8-ART124
https://doi.org/10.1145/3731189

Online free-view navigation in volumetric videos requires high-quality rendering and real-time streaming in order to provide immersive user experiences. However, existing methods (*e.g.*, dynamic NeRF and 3DGS) may not handle dynamic scenes with complex motions, and their models may not be streamable due to storage and bandwidth constraints. In this paper, we propose a novel 4D Gaussian Video (4DGV) approach that enables the creation and streaming of photorealistic, volumetric videos for dynamic scenes over the Internet. The core of our 4DGV is a novel streamable group of Gaussians (GOG) representation based on motion layering. Each GOG consists of static and dynamic points obtained via lifting 2D segmentation into 3D in motion layering, where the deformation of each dynamic point is represented as the temporal offset of its attributes. We also adaptively convert static points back to dynamic points to handle the appearance change, (*e.g.*, moving shadows and reflections), of static objects through optimization. To support real-time streaming of 4DGVs, we show that by applying quantization on Gaussian attributes and H.265 encoding on deformation offsets, our GOG representation can be significantly compressed (to around 6% of the original model size) without sacrificing the accuracy (PSNR loss less than 0.01dB). Extensive experiments on standard benchmarks demonstrate that our method outperforms state-of-the-art volumetric video approaches, with superior rendering quality and minimum storage overheads.

CCS Concepts: • **Computing methodologies → Dynamic Reconstruction**.

Additional Key Words and Phrases: Volumetric Video, Motion Layering, Group of Gaussians, Gaussian Video Compression

ACM Reference Format:
Pinxuan Dai, Peiquan Zhang, Zheng Dong, Ke Xu, Yifan Peng, Dandan Ding, Yujun Shen, Yin Yang, Xinguo Liu, Rynson W.H. Lau, and Weiwei Xu. 2025. 4D Gaussian Videos with Motion Layering. *ACM Trans. Graph.* 44, 4, Article 124 (August 2025), 18 pages. https://doi.org/10.1145/3731189

1 INTRODUCTION

Volumetric videos [Broxton et al. 2020; Li et al. 2022b] are an emerging form of media that have wide applications ranging from remote presence to holographic classrooms. The key feature of volumetric videos is that they encode intricate 4D space-time scene information for immersive user experiences. Therefore, pursuing high-quality, real-time rendering combined with compact representations for efficient transmission is paramount in developing online volumetric video systems.

Existing volumetric video methods primarily rely on dynamic neural radiance field (NeRF) [Cao and Johnson 2023; Li et al. 2022b; Park et al. 2021] or 3D Gaussian splatting (3DGS) [Kerbl et al. 2023]. In order to handle dynamic scenes, deformable GS-based methods [Wu et al. 2024a; Yang et al. 2024a,b] are recently proposed to enhance the conventional static 3DGS [Kerbl et al. 2023] by allowing Gaussian point attributes to evolve over time. This is accomplished by predicting temporal deformation fields through implicit functions [Wu et al. 2024a; Yang et al. 2024a], or by slicing explicit 4D Gaussian kernels [Yang et al. 2024b] along the temporal axis. While these methods can support real-time rendering and deliver remarkable visual results, they face challenges when deployed online for long videos or scenes featuring complex motions: (1) **Limited Scalability.** The representation capability of implicit neural deformation fields [Wu et al. 2024a; Yang et al. 2024a] is constrained by their fixed model size, restricting their ability to effectively handle lengthy videos and complex motions. (2) **Upsurging Storage Demands.** 3DGS imposes high demands on storage due to the large number of explicit attributes associated with each point. Extending 3DGS with temporal dimension [Duan et al. 2024; Yang et al. 2024b] will introduce more severe storage challenges as more points are required to represent dynamic scenes. Consequently, achieving top-notch, real-time rendering and streamable transmission for volumetric video applications remains a great challenge.

To tackle these challenges, this paper develops a scalable and compact representation for online volumetric videos. The key concept is to integrate the Group of Pictures (GOP) structure in MPEG standard [Sullivan et al. 2012; Sze et al. 2014] as a basic component for efficiently compressing and transmitting volumetric video data. This approach offers dual benefits. First, segmenting the video into groups keeps the memory and computational complexities at manageable levels, providing a scalable solution for handling long volumetric videos. Second, the group structure helps break down complex temporal motions into piecewise functions, reducing the complexity of motion modeling. At the beginning frame of each group (*i.e.*, keyframe), we can adjust the distribution of the Gaussian points to prevent error accumulation in motion tracking. Meanwhile, we observe that a large portion of the scene in a video is typically static. This motivates us to adopt motion layering to separate the dynamic and static Gaussian points, thereby reducing the computational and storage costs associated with learning the deformation of static points.

The main technical contribution of this paper is a novel group of Gaussians (GOG) representation for streamable volumetric videos, which are termed 4D Gaussian Videos (4DGVs). Each GOG consists of dynamic and static points, where dynamic points are represented as canonical points with time-dependent deformation offsets [Yang et al. 2024a] to model the scene motions within a time window. To efficiently reuse the static points across multiple GOGs, we retain their attributes without updates during optimizations. The dynamic/static point separation in 3D space is achieved by the adaptive motion layering step. We first detect raw motion masks for dynamic regions in input frames using optical flow [Wang et al. 2024a], and then leverage semantic segmentation [Ravi et al. 2024b] to complete and refine the motion masks. The 2D motion masks are lifted into Gaussian points by optimizing a motion label parameter associated with each point through the differentiable Gaussian rasterization. To mitigate the impact of inaccurate 2D motion masks and, more importantly, to capture appearance changes like moving reflections and shadows on static objects, we adaptively convert static points to dynamic ones throughout the optimization by tracking the view-space positional gradients of the static points.

Furthermore, we design a progressive frame-sampling strategy to improve the accuracy of the optimized offsets for the dynamic points. During the optimization of deformation offsets, we first sample the frames within a group close to the keyframe in the early training iterations to warm up the deformation field learning, and then the frames that are farther from the keyframe. This strategy avoids the difficulty in predicting long-range motions for the initialized Gaussian points at the early stage of the optimization, which improves the rendering quality of objects with large motions significantly.

Finally, we investigate compression techniques for real-time 4DGV streaming. We apply vector and bit quantization to Gaussian point attributes for point-cloud compaction, and H.265 encoding [Sullivan et al. 2012; Sze et al. 2014] for the compression of point deformation offsets extracted from the learned deformation fields, achieving a compression ratio of 6% with minimal loss in rendering quality (less than 0.01 dB).

Based on GOG and the technical components mentioned above, we have developed a 4DGV system that is capable of streaming volumetric video over the Internet. Extensive experiments on multiple public datasets verify the effectiveness of our proposed 4DGV method, against the existing dynamic scene reconstruction methods in terms of view synthesis quality and storage efficiency, especially for handling complex and fast motions, as illustrated in Fig. 1.

2 RELATED WORK

2.1 Volumetric Video

Creating volumetric videos for dynamic scenes has been a long-standing research focus in the graphics and vision communities. Early studies pursued this objective through various methodologies, including view interpolation [Jain and Wakimoto 1995; Zitnick et al. 2004], dynamic lumigraph [Goldlücke et al. 2002; Schirmacher et al. 2001], scene flow [Vedula et al. 2000; Zhang and Kambhamettu 2001], and geometric cues (*e.g.*, depth and textured mesh) [Broxton et al. 2020; Newcombe et al. 2015]. More recent approaches utilize neural radiance fields (NeRFs) [Li et al. 2022b; Lin et al. 2022; Park et al. 2021; Pumarola et al. 2020; Song et al. 2023] to render realistic novel view images, albeit at the cost of being slow to train and render. Despite efforts being made on acceleration using, *e.g.*, occupancy fields [Dong et al. 2023; Wang et al. 2024b] and spatial factorization [Cao and Johnson 2023; Fridovich-Keil et al. 2023], achieving real-time rendering is still challenging for these NeRF-based approaches.

3D Gaussian Splatting (3DGS) [Kerbl et al. 2023] is an efficient alternative for reconstructing and rendering dynamic scenes by incorporating implicit deformation fields parameterized by Multilayer Perceptrons (MLPs) [Yang et al. 2024a], K-Planes [Wu et al. 2024a], and hashgrid [Sun et al. 2024; Xu et al. 2024a]. However, the MLPs used in [Yang et al. 2024a] compromise the rendering speed, while the K-Planes [Wu et al. 2024a] may struggle to accurately render scene details. In this context, hashgrid-based methods [Sun et al. 2024; Xu et al. 2024a] strike a balance between rendering quality and computational efficiency. Certain studies [Duan et al. 2024; Yang et al. 2024b] employ 4D Gaussian kernels to model temporal dynamics. While these methods effectively accommodate variable-length videos with complex dynamics by extending the point densification mechanism to the temporal dimension, they incur a significant increase in the number of points and model size, requiring several GBs for few-second reconstructions. More detailed discussions on 3DGS can be found in this survey [Wu et al. 2024b].

TGH [Xu et al. 2024c] introduces a temporal hierarchy structure and achieves implicit static/dynamic separation based on deformation velocity. Given the whole multi-view video as input, it demonstrates impressive performance in long video reconstruction. However, it is difficult to directly apply video compression techniques to their representation since its data organization, without post-processing, is not compatible with the group structure used in video compression. In contrast, our representation is intentionally designed and restructured based on the group structure. It is more friendly to video compression and able to achieve high-quality reconstruction results even after compression. Furthermore, TGH requires per-frame sparse points for initialization, whereas our method only necessitates sparse points in the first frame and progressively reconstructs the scene dynamics using sequential GOGs.

There are also research works on reconstructing streamable volumetric videos using 3DGS. 3DGStream [Sun et al. 2024] models a dynamic scene using per-frame deformation fields, without considering temporal coherence in optimization. STG [Li et al. 2024] directly divides the input video into multiple groups and models them independently, causing storage redundancy and flicker artifacts during group transitions. In contrast, our method integrates incremental reconstruction and a fade-out/fade-in operation at group boundaries to remove the flickering. Ex4DGS [Lee et al. 2024c] separates the points into fully dynamic points and linearly moving points in the explicit Gaussian attribute interpolation. In contrast, we assume that static points do not move, and they only need to be stored once and shared across multiple groups to further reduce the storage.

2.2 Motion Layering Methods

Optical flow plays a crucial role in motion layering methods [Brox and Malik 2010; Hui et al. 2018; Janai et al. 2018; Ling et al. 2022; Ren et al. 2019; Teed and Deng 2020; Tian and Andrade-Cetto 2024; Weinzaepfel et al. 2013; Zhu et al. 2024]. Based on 4D correlation volume and iterative refinement, optical flow-based approaches excel at accurately estimating local motions. However, these methods often struggle to effectively handle long-range motions. This issue may be mitigated via tracking-based methods [Doersch et al. 2022, 2023; Harley et al. 2022; Karaev et al. 2024; Ravi et al. 2024a; Sand and Teller 2008].

Another line of methods model motions by performing a global optimization across the entire video [Chang et al. 2013; Pumarola et al. 2020; Rubinstein et al. 2012; Sun et al. 2010; Vondrick et al. 2018]. Typical methods include omnimattes [Gu et al. 2023; Lin et al. 2023; Lu et al. 2021; Suhail et al. 2023] and deformable NeRF/GS [Weng et al. 2022; Yang et al. 2024a; Zhu et al. 2024]. OmniMotion [Wang et al. 2023a] leverages NeRF [Mildenhall et al. 2021; Pumarola et al. 2020] to provide 3D supervision while incorporating a normalizing flow network [Sun et al. 2010] to model deformations. FastOmniTrack [Song et al. 2024] extends OmniMotion by integrating depth guidance [Bhat et al. 2023] and DINOv2 feature [Oquab et al. 2023]. MotionGS [Zhu et al. 2024] introduces a multi-flow approach to guide the deformation of 3D Gaussians. Lee *et al.* [2024a] propose to combine the generative prior of video diffusion models for motion layering.

In this work, we extend 2D motion layering by lifting onto 3D space for explicit initialization of dynamic points, and adaptively convert static points to dynamic ones as needed. This design helps to reduce both computation and storage overheads during the reconstruction for dynamic scenes.

2.3 Point Cloud Compression

Point clouds are a widely used 3D representation method renowned for their ability to faithfully capture the intricate details of 3D objects. However, they suffer from high storage demands, causing high data transmission and processing overheads. Numerous efforts have been made to develop efficient Point Cloud Compression (PCC) algorithms aimed at minimizing the size of point clouds with a minimal loss. The MPEG standard, G-PCC [ISO 2018a], employs an octree structure for efficient compression of point cloud geometry and attribute. DRACO [Google 2017] encodes the point attributes with the range Asymmetric Numeral System (rANS) [Duda 2014] based on k-D trees. V-PCC [ISO 2018b] iteratively segments a sequence of point clouds into patches, flattens the patches onto 2D images, and utilizes existing video codecs [Kalva 2006; Sullivan et al. 2012; Sze et al. 2014; Wiegand et al. 2003] to efficiently compress flattened point cloud images. However, the patch segmentation easily leads to projection distortions, especially for the irregular and detailed geometry of the 3D Gaussian point clouds.

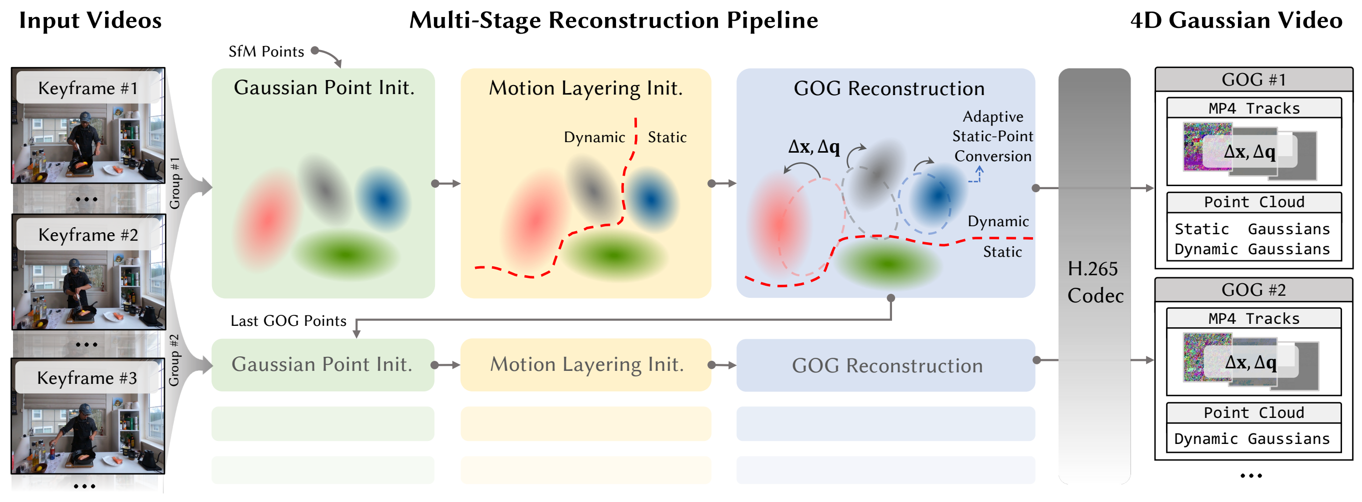


Fig. 2. **4DGV Overview**. We take a multi-view video as input and employ the 4DGV representation to transform each group of input frames into a group of Gaussians (GOG). The process involves the following stages: (a) *Gaussian point initialization* learns to establish static Gaussian point clouds as initialization at each keyframe timestamp. (b) *Motion layering initialization* separates dynamic and static points, enabling static points to be shared across multiple groups. (c) *GOG reconstruction* learns to deform dynamic points for modeling scene dynamics within each frame group. The points deformed to the next keyframe timestamp serve as initialization for the subsequent group. We further leverage the H.265 codec to encode time-dependent point deformations as multiple MP4 tracks for efficient compression. Finally, the reconstructed GOGs are consolidated for seamless real-time streaming.

While 3D Gaussian representation can be regarded as a point cloud with geometry coordinates and a set of attributes, substantial efforts have been devoted for efficient compression [Kerbl et al. 2023]. Some methods aim to reduce redundant points using structured scaffolding [Chen et al. 2024; Lu et al. 2024] and densification strategies [Fan et al. 2023; Mallick et al. 2024]. Others focus on compressing the spherical harmonics (SH) appearance model using distillation [Fan et al. 2023], appearance grid [Chen et al. 2024; Lee et al. 2024b], vector quantization [Navaneet et al. 2024; Niedermayr et al. 2024; Wang et al. 2024d], and dynamic expansion for SH degrees [Xu et al. 2024c]. Most recently, V^3 [Wang et al. 2024c] adopts the H.265 codec to compress dynamic Gaussian point attributes, and achieves real-time streaming for human performance rendering. However, this method leads to relatively large compression loss as revealed in our experiments. In this work, we show that the H.265 codec is more effective when applied to point deformations, resulting in lower compression-induced losses.

3 PRELIMINARY

Our method is built upon the recent techniques of 3DGS [Kerbl et al. 2023] and its extensions to dynamic scenes [Yang et al. 2024a]. 3DGS employs an explicit Gaussian point cloud for scene representation, with each Gaussian point characterized by a set of attributes $\{\mathbf{x}_i, \mathbf{q}_i, \mathbf{s}_i, o_i, \mathbf{c}_i\}_{i\in\mathcal{P}}$, which are its center position, rotation, scale, opacity and color features represented by spherical harmonic coefficients, respectively, with i being the point index. The Gaussian kernel weight α_i is computed from these attributes. The color of each pixel is rendered by alpha-blending point colors along the ray:

$$\tilde{C} = \sum_{i=0}^{n} T_i \alpha_i \mathbf{c}_i, \qquad T_i = \prod_{j=0}^{i-1} (1 - \alpha_j). \tag{1}$$

The deformable 3DGS [Yang et al. 2024a] methods typically learn temporal-varying deformation offsets of Gaussian points via implicit deformation fields parameterized by neural networks Θ:

$$\Delta\mathbf{x}_{i,t}, \Delta\mathbf{q}_{i,t}, \Delta\mathbf{s}_{i,t} = \mathbf{F}_\Theta(\mathbf{x}_i, t),$$

where Δ indicates the offset, and i and t are the indexes of Gaussian points and timestamps, respectively. Some other methods also let the $o_{i,t}$ to vary over time. During optimization, the offsets are applied to Gaussian points in the canonical space such that their rendering results align with each video fame. The Gaussian points in canonical space are jointly optimized with the deformation network.

4 4D GAUSSIAN VIDEO

Given a multi-view variable-length $t\in[0, +\infty)$ video stream as the input $\{I_{v,t}\}$, where $v\in\mathcal{V}$ is the view index, we designate a keyframe every 30 frames and obtain frame groups $\{I_{v,t}\}_{v\in\mathcal{V}, t\in[k_i, k_{i+1})}$ accordingly, where k_i indicates the keyframe timestamp. Our 4D Gaussian Video (4DGV) system reconstructs groups of Gaussians $\text{GOG}_k = \{\mathcal{P}^s_k, \mathcal{P}^d_k, \mathcal{D}_k\}$ from frame groups, where $\mathcal{P}^s_k$ and $\mathcal{P}^d_k$ represent the sets of static and dynamic points, respectively, and $\mathcal{D}_k$ indicates the set of deformation offsets for $\mathcal{P}^d_k$.

As shown in Fig. 2, we reconstruct GOGs in a 4DGV through a multi-stage reconstruction pipeline, and encode them using the H.265 codec. Within this pipeline, the initialization of Gaussian points and motion layering are applied to keyframe timestamps k_i,

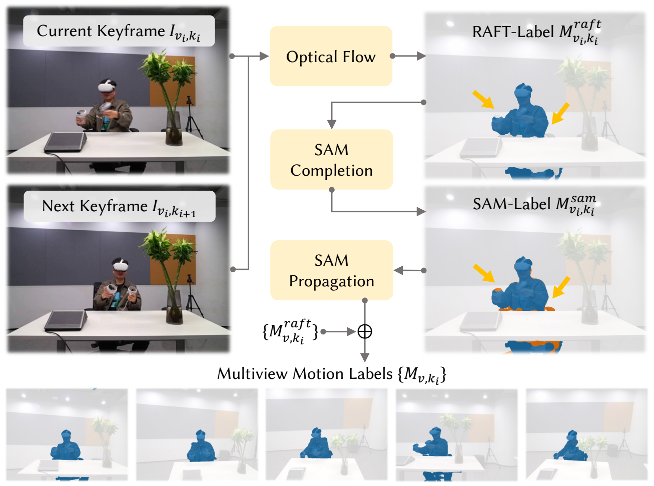

Current Keyframe I_{v_i,k_i}
Next Keyframe $I_{v_i,k_{i+1}}$
Optical Flow
SAM Completion
SAM Propagation
RAFT-Label $M^{raft}_{v_i,k_i}$
SAM-Label $M^{sam}_{v_i,k_i}$
$\{M^{raft}_{v,k_i}\}$
Multiview Motion Labels $\{M_{v,k_i}\}$

Fig. 3. **Motion label image generation.** Pixels with a dynamic label in images are highlighted with colors. We use optical flow [Wang et al. 2024a] to compute RAFT-label $M^{raft}_{v_i,k_i}$ (highlighted in blue), and use SAM2 [Ravi et al. 2024b] to fill in the missing parts of the dynamic object in $M^{sam}_{v_i,k_i}$ (highlighted in orange).

providing an initial Gaussian point cloud $\{\mathcal{P}^s_k, \mathcal{P}^d_k\}$ with dynamic-static separation (Sec. 4.1). The initial Gaussian points at the first frame of the video are obtained through the standard 3DGS pipeline. Subsequently, $\mathcal{P}^d_k$ is deformed to model the scene motion within each frame group (Sec. 4.2). We fix the attributes of $\mathcal{P}^s_k$ during the GOG reconstruction and share $\mathcal{P}^s_k$ across groups to substantially reduce the optimization and storage costs in GOG, while allowing $\mathcal{P}^d_k$ to be optimized, pruned, and cloned as in 3DGS. Furthermore, we deform the Gaussian points in previous GOG to the next keyframe as a dynamic-point initialization for the next group to incrementally reconstruct the whole video. It can reduce the initialization time at each group and is beneficial in reconstructing a long-duration volumetric video.

Finally, we incorporate the quantization and compression to reduce the model size (Sec. 4.3).

4.1 Adaptive Motion Layering

The motion layering starts with initializing a motion label, *i.e.*, static or dynamic, for each pixel in the multi-view images of the keyframes using 2D vision models [Ravi et al. 2024b; Wang et al. 2024a]. These motion labels are then lifted to Gaussian points through differentiable Gaussian rasterization. Due to the attributes of static points are fixed during the latter GOG reconstruction, an adaptive static-point conversion is applied to convert some static points to dynamic ones. The converted points are used to capture the appearance changes in static objects.

2D Motion Segmentation. This step aims to obtain the corresponding motion label images $\{M_{v,k}\}_{v\in\mathcal{V},k\in\mathcal{K}}$ at the keyframes $\mathcal{K}$. Due to the static camera setting in the multi-view capture, it is straightforward to classify a pixel to be static if its flow vector is small or 0. To this end, on each keyframe k_i, we apply the RAFT model [Wang et al. 2024a], $\mathbf{F}_{raft}$, to estimate the optical flow between the multi-view images of the current keyframe and those of the next keyframe. If the length of the flow is larger than λ_{raft} pixels, we label it as dynamic; otherwise, we label it as static. A motion label image can then be obtained as follows:

$$M^{raft}_{v,k_i} = \left\|\mathbf{F}_{raft}(I_{v,k_i}, I_{v,k_{i+1}})\right\| > \lambda_{raft}.$$

However, we find that raft motion labels $\{M^{raft}_{v,k}\}$ may fail to cover slow-moving parts of the dynamic objects in the scene, such as the chair and the controller shown in Fig. 3. When one part of an object is classified as static and another part as dynamic, it can result in tearing artifacts in dynamic objects. Thus we enforce object-level consistency for the motion layering labels. We select a view v_i to sample points from $M^{raft}_{v_i,k_i}$ to query the SAM2 model [Ravi et al. 2024b], $\mathbf{F}_{sam}$, and obtain the object-level SAM-labels $M^{sam}_{v_i,k_i}$ for completion. Utilizing the memory attention mechanism in SAM2, the $M^{sam}_{v_i,k_i}$ for one view can be efficiently propagated to all other views, obtaining multiview-consistent SAM-labels $\{M^{sam}_{v,k_i}\}_{v\in\mathcal{V}}$. The motion labels for each view are formed by combining the RAFT- and SAM-labels:

$$M^{sam}_{v_i,k_i} = \mathbf{F}_{sam}(M^{raft}_{v_i,k_i}), \qquad M_{v,k_i} = M^{raft}_{v,k_i} + M^{sam}_{v,k_i}.$$

3D Lifting. We lift the 2D motion labels into 3D points by associating a single-channel point motion label m_i to the i_{th} Gaussian point and rendering each ray into 2D image pixel $\tilde{M}$ using alpha blending (same as Eq. 1): $\tilde{M} = \sum_{i=0}^{n} T_i\alpha_i m_i$. The per-Gaussian motion label m_i is optimized through the standard differentiable Gaussian rasterization pipeline by minimizing the loss, as:

$$\mathcal{L}_m = \sum_{v\in\mathcal{V}} |\tilde{M}_v - M_v| + \lambda^{nn}_{seg} \sum_{i\in\mathcal{P}} |m_i - m_{nn}| + \lambda^{dyn}_{seg} \sum_{i\in\mathcal{P}} |m_i - 1|,$$

where the subscript nn is the nearest neighbor point index, λ^{nn}_{seg} and λ^{dyn}_{seg} are two balancing hyper-parameters. The first term supervises the rendered point motion labels $\tilde{M}_v$ using the generated motion labels M_v at the corresponding views, where the group keyframe index k is discarded for simplicity. The second term improves noise resistance by enforcing local similarity of the point motion labels. The last term encourage the points to have dynamic label, which is effective when the multiview 2D labels are inconsistent.

Adaptive Static-Point Conversion. The lifted motion labels for static objects are usually semantically correct. However, Since we fixed the attributes of the static points in the GOG reconstruction, the appearance change that occurs in some regions of the static objects, such as moving shadows and view-dependent reflections, may not be captured well. Fig. 4 illustrates a moving shadow example. To address this problem, we propose converting static points to dynamic points if: (1) their screen-space positional gradients exceed λ^{grad}_{adp} or (2) the proportion of dynamic points among their nearest neighbors is greater than λ^{nn}_{adp}. As a result, the attributes of the Gaussian points in these regions can be optimized and cloned to reduce the rendering errors. This adaptive layering strategy expands the layering results from 3D lifting, improving the layering flexibility and reconstruction quality. We observe that it also improves the

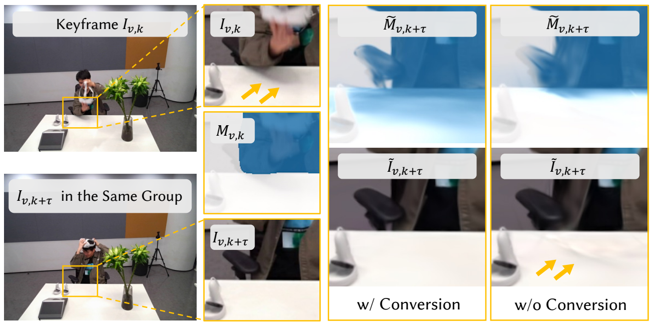


Fig. 4. **Adaptive static-point conversion.** Pixels with a dynamic label in images are highlighted in blue, and with a static label in white. Although the motion labels $M_{v,k}$ for view v at keyframe k are semantically correct, the layered static points on the table fail to model the moving shadow. This causes artifacts in $\tilde{I}_{v,k+\tau}$. The adaptive conversion mitigates the artifacts by extending the motion labels $\tilde{M}_{v,k+\tau}$ to cover the intended areas.

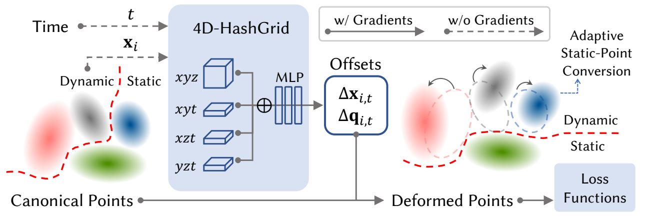


Fig. 5. **The GOG reconstruction** We employ the 4D-hashgrid [Xu et al. 2024a] to model the deformation of the layered dynamic points, and correct false-static points during optimization.

reconstruction of newly appearing objects and previously occluded areas, which are often under-reconstructed due to their invisibility in keyframes. The adaptive conversion is performed every 100 iterations during the optimization in the GOG reconstruction, which will be discussed in Section 4.2.

4.2 The GOG Reconstruction

Our goal is to jointly optimize the dynamic points in canonical space $\mathcal{P}_k^d$, and the deformation offsets of dynamic points $\mathcal{D}_k$ in this stage as shown in Fig. 5. For simplicity, the group index k is discarded hereafter. The set $\mathcal{D}$ contains two kinds of deformation offsets in our system: position offset $\Delta\mathbf{x}$, and rotation offset $\Delta\mathbf{q}$, since we observe no performance improvements by allowing point scale and opacity to change over time in our experiments. The predicted deformation offsets transform the i_{th} Gaussian point in the canonical space to corresponding frame t as:

$$\mathbf{x}_{i,t} = \mathbf{x}_i + \Delta\mathbf{x}_{i,t}, \qquad \mathbf{q}_{i,t} = \mathbf{q}_i \cdot \Delta\mathbf{q}_{i,t}.$$

We parameterize the deformation fields with a 4D-hashgrid [Xu et al. 2024a] considering its faster convergence over MLPs [Yang et al. 2024a] and the larger representation capacity over K-Planes [Wu et al. 2024a]. The 4D-hashgrid consists of 1 spatial (xyz) and 3 spatial-temporal (xyt, xzt, yzt) hashgrids. We set the t-axis at a relatively lower resolution due to the temporal smoothness of the deformations. The concatenated grid features from four grids are then decoded into deformation offsets by multi-head tiny MLPs.

Loss Functions. The GOG reconstruction is optimized by the following loss terms: photometric loss L_{rgb}, as-rigid-as-possible (ARAP) loss L_{arap}, and temporal smoothness loss L_{temp}. The photometric loss is calculated between the rendered image $\tilde{I}$ and the ground truth video frames I:

$$\mathcal{L}_{rgb} = (1-\lambda)\left|I - \tilde{I}\right| + \lambda L_{dssim}(I, \tilde{I}).$$

Inspired by [Duan et al. 2024; Huang et al. 2024], we employ regularization to encourage spatial-temporal local smoothness in motion modeling. First, as we expect the adjacent dynamic points to share similar deformation at each deformation timestamp, we apply the ARAP loss as:

$$\begin{aligned}\mathcal{L}_{arap} &= \left|\Delta\mathbf{q}_{i,t} - \Delta\mathbf{q}_{nn,t}\right| + \left|\Delta\boldsymbol{\tau}_{i,t} - \Delta\boldsymbol{\tau}_{nn,t}\right| \\ &\quad + \left|\left\|\mathbf{x}_i - \mathbf{x}_{nn}\right\| - \left\|\mathbf{x}_i + \Delta\mathbf{x}_{i,t} - \mathbf{x}_{nn} - \Delta\mathbf{x}_{nn,t}\right\|\right|, \\ \Delta\boldsymbol{\tau}_{i,t} &= \mathbf{x}_i + \Delta\mathbf{x}_{i,t} - \Delta\mathbf{R}_{i,t} \cdot \mathbf{x}_i^{\top},\end{aligned}$$

where $\Delta\boldsymbol{\tau}$ denotes the point translation offset, the subscript nn is the index of the nearest neighbor point, $\Delta\mathbf{R}$ is the matrix form of $\Delta\mathbf{q}$, and $\mathbf{x}$ is a row vector. While the first and second terms encourage consistent rotation and translation offsets between neighbors, the third term maintains the local metric length. Second, to force the deformation of each point to be linearly smooth during a short time window $[t, t']$ in the temporal dimension, we apply $\mathcal{L}_{temp}$ as:

$$\begin{aligned}\mathcal{L}_{temp} &= \left|(1-\delta)\Delta\mathbf{x}_{i,t} + \delta\Delta\mathbf{x}_{i,t'} - \mathbf{x}_{i,t+\delta(t'-t)}\right| \\ &\quad + \left|(1-\delta)\Delta\mathbf{q}_{i,t} + \delta\Delta\mathbf{q}_{i,t'} - \mathbf{q}_{i,t+\delta(t'-t)}\right|,\end{aligned}$$

where $\delta \in (0,1)$ is a random perturbation. This temporal smooth loss $\mathcal{L}_{temp}$ encourages constant velocity for both the point position and rotation. The whole optimization loss is formed as:

$$\mathcal{L} = \mathcal{L}_{rgb} + \lambda_{arap}\mathcal{L}_{arap} + \lambda_{temp}\mathcal{L}_{temp},$$

where λ_{arap} and λ_{temp} are two balancing hyper-parameters.

Progressive Frame Sampling. During the optimization of the GOG reconstruction, we progressively sample frames from near to far during the first N training iterations. This helps warm up the deformation field learning and improves long-range motion reconstruction as demonstrated in Fig. 9. The possibility of timestamp t being sampled in iteration $n \in (0, N]$ is:

$$p(n,t) = \frac{p'(n,t)}{\sum_{\tau=0}^{T} p'(n,\tau)}, \qquad p'(n,t) = \begin{cases} 1 & \text{if } t \le \lfloor T \cdot n/N \rfloor \\ 0 & \text{else} \end{cases},$$

where T is the group length. We set N to half of the total training iterations, and evenly sample timestamps in the remaining iterations.

4.3 The GOG Compression

In this section, we describe how to apply vector quantization, bit quantization, and H.265 encoding to compress GOG, significantly reducing its storage to allow online transmission at the cost of minimally reduced rendering quality.

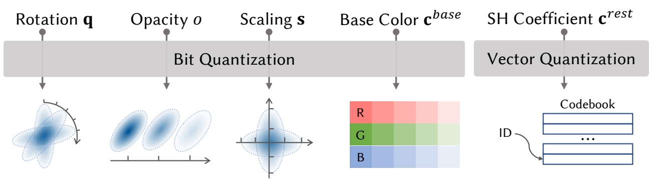


Fig. 6. **Gaussian point quantization.** We use bit quantization to discretize the floating point values of $\{\mathbf{q}, o, \mathbf{s}, \mathbf{c}^{base}\}$, and apply vector quantization to efficiently store high-order SH coefficients $\mathbf{c}^{rest}$ in a small codebook.

Gaussian Point Quantization. As shown in Fig. 6, we use different quantization techniques for static and dynamic Gaussian points in the canonical space. We first quantize their 32-bit Gaussian point attributes in `float` format $\{\mathbf{q}, o, \mathbf{s}, \mathbf{c}_{base}\}$ using fewer bits to reduce the model size. 8 bits are used for attributes with a small range of values, including $\mathbf{q}$, o, and $\mathbf{s}$. For $\mathbf{s}$, we evenly discretize the value in the range of $[2^{-n\_bit}, 1]$ times the scene scale. In practice, we find that $n\_bit = 12$ is sufficient to quantize $\mathbf{s}$ for medium-sized scenes. The ranges of the quantized attributes are all recorded for recovery.

Second, the 3rd-order SH used in 3DGS [Kerbl et al. 2023] corresponds to 48 `float` coefficients, which imposes a high storage requirement for the Gaussian point cloud. While the 3-order color features can be divided into two parts: the base color $\mathbf{c}^{base} \in \mathbb{R}^3$ and the high-order coefficients $\mathbf{c}^{rest} \in \mathbb{R}^{45}$, we empirically find that $\mathbf{c}^{rest}$ accounts for 94% of the storage in the color representation but contributes only ~15% to the outgoing radiance energy. Hence, we apply vector quantization to $\mathbf{c}^{rest}$ to significantly reduce the model size. Specifically, we cluster similar $\mathbf{c}^{rest}$ and compute the corresponding cluster centers for rendering. An integer cluster index is assigned to each Gaussian point to locate its closest cluster center in a small codebook of size N_{vq}. Our experiments reveal that $N_{vq} = 1024$ is enough to produce high-quality rendering results.

Deformation Encoding. Since the deformation offsets of points (comprising $\Delta\mathbf{x}$ and $\Delta\mathbf{q}$) are time-varying high-dimensional offset features, we apply H.265 video compression codec [Sullivan et al. 2012; Sze et al. 2014] to the offsets by organizing them into 2D offset maps at each timestamp using Morton sorting [Morton 1966]. The deformation offsets within a group are typically in a small range, making them highly resilient to bit quantization in H.265 encoding.

After Morton sorting using deformation offsets, each dynamic point is assigned with a 2D pixel position, efficiently mapping the 3D point cloud structure to the 2D image space. The deformation offsets are stored in the corresponding pixel, forming the offset maps containing 3-channel $\Delta\mathbf{x}$ and 4-channel $\Delta\mathbf{q}$. As $\Delta\mathbf{x}$ is crucial for preserving the rendering quality, we follow [Wang et al. 2024c] to quantize it using 16 bits and then splitting its high and low halves into two 8-bit RGB images. For $\Delta\mathbf{q}$, we use 8-bit grayscale images to quantize each of its channels and concatenate the four images into one.

The quantized offset images are then compressed using the H.265 codec into three 8-bit MP4 tracks: high-order $\Delta\mathbf{x}$, low-order $\Delta\mathbf{x}$, and the concatenated $\Delta\mathbf{q}$. The high-order $\Delta\mathbf{x}$ is encoded losslessly and the latter two are compressed using a Constant Rate Factor (CRF) value of 20.

4.4 Real-Time Rendering

The rendering of 4DGV on each timestamps follows the Gaussian rasterization pipeline in 3DGS [Kerbl et al. 2023]. We deform the dynamic points in canonical space $\mathcal{P}_k^d$ to the corresponding timestamp by applying the deformation offsets $\mathcal{D}_k$. The deformed dynamic points are concatenated with static points $\mathcal{P}_k^s$ to render the complete scene. To smooth out sudden changes during group transitions, we introduce an overlapping window between groups for seamless rendering. Within this window, we linearly scale the opacity of $\mathcal{P}_k^d$ from one to zero for fade-out, while simultaneously scaling the opacity of $\mathcal{P}_{k+1}^d$ from zero to one for fade-in.

We have also implemented an online web player for 4DGV based on WebGL, which supports real-time streaming playback over the Internet using PCs, pads and mobile phones. However, since the parallel prefix sum used in CUDA implementation in 3DGS is not available in WebGL, we rely on converting each Gaussian point into a quadrilateral for the rasterization using shaders as in Splat [Kwok 2023]. It results in slightly different rendering results in the web player, comparing to the CUDA-based Gaussian rasterization pipeline.

5 EXPERIMENTS

We implement our 4DGV approach and conduct all experiments on a single NVIDIA RTX 4090 GPU, achieving state-of-the-art rendering quality while significantly reducing the model size. As an example, our method achieves an average 4DGV video size of 21.2 MB on Neu3DV [Li et al. 2022b] dataset, and the videos contain initial points of 3.6 MB for the first frame and streamable GOGs of 1.76 MB per second on average. Moreover, the adaptive motion layering labels 25.4% points as dynamic on this dataset, reducing the computation and storage cost of the deformations by 4×. We report comparisons, compression performance, and ablation studies in the following. More details and results can be found in the supplementary file and video.

5.1 Implementation Details

For the group initialization, we train the first group with 20k iterations, while subsequent groups utilize 5k for fast refinement. The 3D lifting of motion labels are trained for 400 iterations per group, and the GOG reconstruction with deformation is trained for 400 iterations per frame. While most hyperparameters from Gaussian rasterization [Kerbl et al. 2023] and 4D-hashgrid [Xu et al. 2024a] have been retrained, we have made adjustments by increasing the opacity culling threshold to 0.05 to suppress floaters and a lighter MLP deformation decoder for improved computational efficiency. The default length of frame groups is set to 30 and use 5-frame overlapping window between groups for smooth transition. Other hyperparameter settings are detailed in the supplementary material.

5.2 Evaluation Settings

Evaluation Datasets. We have conducted evaluations of our method on three public datasets for multiview dynamic reconstruction: Neu3DV [Li et al. 2022b], MeetRoom [Li et al. 2022a], and Technicolor [Sabater et al. 2017]. Our evaluation follows established evaluation conventions, except using all available frames in the Technicolor dataset instead of limiting the evaluation to 50-frame

Table 1. **Quantitative comparisons on the MeetRoom [Li et al. 2022a], Technicolor [Sabater et al. 2017], and our DeskGames datasets.** We report the PSNR, SSIM, and LPIPS metrics, along with the model size (in MB), to evaluate both rendering quality and storage efficiency. The reported scores are computed using their released code.

Methods	MeetRoom				Technicolor				DeskGames			
	PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓
Grid4D [Xu et al. 2024a]	30.65	0.948	0.163	297	30.37	0.888	0.210	378	26.29	0.863	0.228	334
STG [Li et al. 2024]	29.51	0.932	0.209	71.4	32.35	0.905	0.186	436	28.35	0.902	0.190	396
3DGStream [Sun et al. 2024]	30.88	0.947	0.186	2287	25.48	0.708	0.446	2615	30.51	0.917	0.173	2243
Ex4DGS [Lee et al. 2024c]	31.03	0.946	0.186	75.0	31.33	0.888	0.229	365	29.48	0.906	0.180	386
4DGS [Yang et al. 2024b]	31.94	0.953	0.177	2196	28.92	0.798	0.347	3070	30.11	0.905	0.199	2616
V^3 [Wang et al. 2024c]	26.12	0.864	0.317	81.8	25.08	0.746	0.340	199	26.05	0.830	0.267	225
Ours	32.31	0.957	0.173	19.2	31.77	0.893	0.197	40.2	30.89	0.920	0.168	34.5

Table 2. **Quantitative comparisons on the Neu3DV [Li et al. 2022b] datasets with 14 baselines.** We report the PSNR, model size (in MB), reconstruction time (in mins), and rendering FPS on average. The metrics are extracted from original papers. *: metrics only on the *flame_salmon* scene.

Methods	PSNR	Size	Time	FPS
DyNeRF* [Li et al. 2022b]	29.58	28	80640	0.015
StreamRF [Li et al. 2022a]	28.26	9420	75	8
NeRFPlayer [Song et al. 2023]	30.69	5134	3600	0.05
HyperReel [Attal et al. 2023]	31.10	360	N/A	2.0
K-Planes [Fridovich-Keil et al. 2023]	31.63	311	108	N/A
MixVoxels-X [Wang et al. 2023d]	31.73	500	300	4.6
MSTH [Wang et al. 2023b]	32.37	135	20	15
4DGaussians [Wu et al. 2024a]	31.15	90	40	30
Grid4D [Xu et al. 2024a]	31.49	N/A	N/A	N/A
STG [Li et al. 2024]	32.05	200	100	140
Ex4DGS [Lee et al. 2024c]	32.11	115	36	121
4DGS [Yang et al. 2024b]	32.01	N/A	N/A	114
4DRotorGS [Duan et al. 2024]	31.62	N/A	60	277
3DGStream [Sun et al. 2024]	31.67	2430	60	215
Ours	32.55	21	73	185

Table 3. **Quantitative comparisons on long videos**. We report the PSNR, and model size (in MB). Metrics are collected from TGH [Xu et al. 2024c], except for ours. *: per-frame reconstruction.

Methods	Neu3DV (1200 frames)		MobileStage (1600 frames)		ENeRF-Outdoor (1200 frames)	
	PSNR ↑	Size ↓	PSNR ↑	Size ↓	PSNR ↑	Size ↓
ENeRF	23.48	830	19.14	1400	25.02	780
3DGS*	28.61	37500	28.02	106000	24.02	115000
4DGS	28.89	2680	23.22	1830	24.64	2450
TGH	29.44	90	27.29	420	24.74	310
Ours	29.26	58	28.21	193	25.81	255

fragments as done in previous methods. As the aforementioned three datasets typically contain only 200–300 frames, we have further evaluated and compared our method on longer video sequences: the *Flame Salmon* scene with 1,200 frames from the Neu3DV dataset, the *Dance3* with 1,600 frames from the MobileStage [Xu et al. 2024b] dataset, and three scenes with 1200 frames from the ENeRF-Outdoor [Lin et al. 2022] dataset.

Furthermore, we have created the DeskGames dataset to establish a more challenging benchmark for evaluation. We use a synchronized multiview system with 21 forward-facing cameras for capturing scenes at a resolution of $2,560 \times 1,440$. The dataset comprises three scenes (*Cube*$^+$, *Domino*$^+$, and *UNO*$^+$), each spanning 6,000 frames and featuring relatively fast motions and frequent interactions between objects. In line with established practices [Li et al. 2022a,b], we have designated the centrally located *cam7* as the test view. Note that 6,000 frames pose a challenge for existing methods, we have extracted four 300-frame clips (*Cube*, *Domino*, *Reading*, and *UNO*) to form a subset for comparison with exisitng mehtods in Table 1. We have reported the results of our method on the complete videos in Table 5.

Baselines and Metrics. We first compare with six latest methods (*i.e.*, Grid4D [Xu et al. 2024a], STG [Li et al. 2024], Ex4DGS [Lee et al. 2024c], 4DGS [Yang et al. 2024b], 3DGStream [Sun et al. 2024], and V^3 [Wang et al. 2024c]) on the MeetRoom [Li et al. 2022a], Technicolor [Sabater et al. 2017], and our DeskGames datasets, by using their official codes and default configurations. Note that we constrain the maximum point number to $3e^6$ for 4DGS [Yang et al. 2024b] to avoid out-of-memory issues on the testing GPU with 24GB VRAM. V^3 [Wang et al. 2024c] is proposed for human performance rendering and use NeuS2 [Wang et al. 2023c] to initialize keyframe point cloud. We initialize it using 3DGS [Kerbl et al. 2023] instead, as NeuS2 cannot handle scene-level reconstruction. We then compare our method with 14 existing volumetric video methods on the widely-used Neu3DV [Li et al. 2022b] dataset. Their results are directly obtained from original papers. Finally, we compare to ENeRF [Lin et al. 2022], per-frame 3DGS [Kerbl et al. 2023], 4DGS [Yang et al. 2024b], and TGH [Xu et al. 2024c] on long video sequences from the Neu3DV, MobileStage [Xu et al. 2024b], and ENeRF-Outdoor [Lin et al. 2022] datasets.

We use three metrics to measure the rendering quality: the Signal-to-Noise Ratio (PSNR), Structure Similarity Index Measure (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). We also report the model size, reconstruction time, and FPS for comparisons. The metrics of each dataset are averaged over all selected scenes.

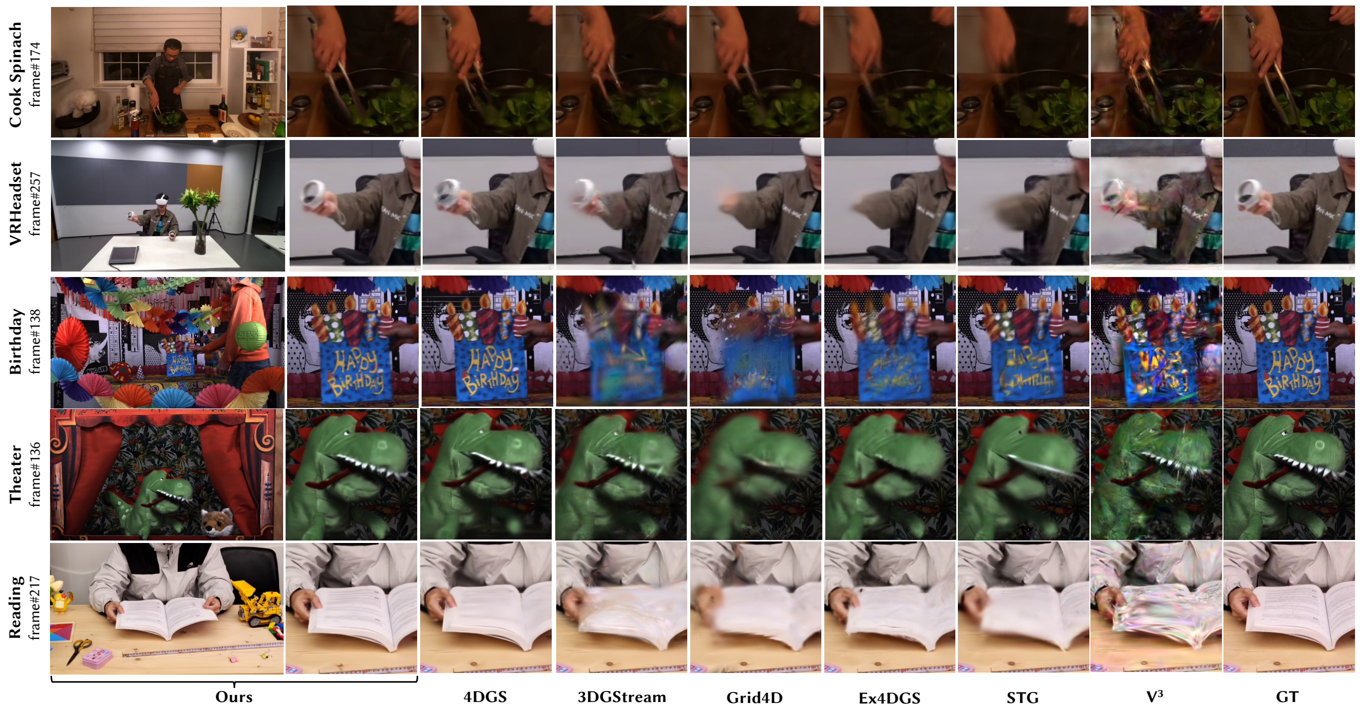


Fig. 7. **Qualitative comparisons on fast motions.** Our method reconstructs objects under challenging fast motions accurately, where previous methods tend to produce blurry results.

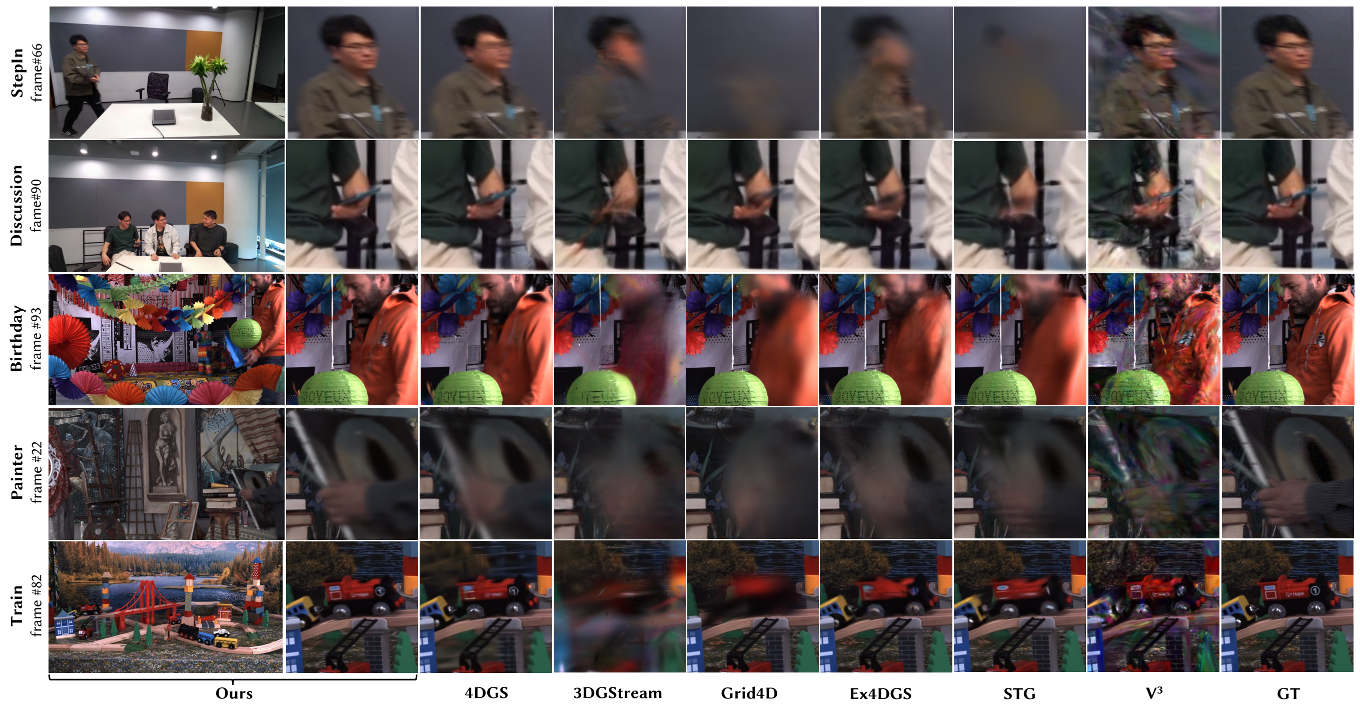


Fig. 8. **Qualitative comparisons on newly appeared objects.** Our method renders newly emerging objects of high quality, while previous methods result in incomplete reconstructions.

Table 4. **Comparison with V^3 [Wang et al. 2024c] on compression**. We report the average PSNR and model size (in MB) on the Neu3DV dataset. Metrics before and after compression are denoted as "w/o Comp." and "w/ Comp." respectively.

Methods	V^3 w/o Comp.	V^3 w/ Comp.	**Ours** w/o Comp.	**Ours** w/ Comp.
PSNR ↑	31.27	25.45	32.56	32.55
Size ↓	4683	113	255	21

Table 5. **Metrics of our method on the full-length DeskGames dataset.** We use the averaged PSNR, SSIM, and LPIPS to measure the rendering quality, and also report the model size (in MB) and the number of points.

Scenes	PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓	Points ↓
Cube$^+$	30.610	0.9185	0.1685	299	7.43M
Dominos$^+$	31.702	0.9294	0.1585	410	7.90M
UNO$^+$	30.550	0.9215	0.1598	675	13.8M

The metrics of our method and V^3 are both from the compressed models unless otherwise specified.

5.3 Main Results

Overall, our method achieves the best performance on the Neu3DV, MeetRoom, and DeskGames datasets and placed second on the Technicolor dataset in terms of novel view synthesis rendering quality as shown in Table 1 and 2. We found that the improvement in rendering quality lies in the more accurate modeling of object motions, especially for:

(1) **Fast Motions.** Our GOG structure and progressive frame sampling strategy improves the reconstruction of fast motions as shown in Fig. 7. We observe 4DGS perform relatively better, while it requires about 100× storage overhead compared to our method. In contrast, our 4DGV approach can reconstruct these objects under fast and large motions accurately.
(2) **Newly Appeared Objects.** As shown in Fig. 8, newly emerging objects transiting from out-of-view to in-view are difficult for existing methods to model due to insufficient observations. Our method produces obviously better reconstruction results for these emerging objects. In our experiments, the re-initialization with the original point densification strategy is able to grow points to explain object emergence, without the need to explicitly add points to expected positions.

We observe that V^3 [Wang et al. 2024c] has successfully reconstructed the complete objects under complex motions, although it is designed for human reconstruction and rendering. However, its renderings exhibit severe artifacts introduced by compression. We hypothesize that this is because V^3 directly quantizes and encodes the point attributes of the entire scene. This compression strategy may performs well for human performance rendering but fails to handle larger scenes effectively. Table 4 compares the performance of V^3 and our method before and after compression. It shows that our choice to encode the point deformation offsets reduces the compression-induced loss in rendering quality.

Table 6. **Ablation on the GOG reconstruction**. We report the average rendering quality, model size (in MB), and compression rate (CR) on the Neu3DV dataset.

Methods	PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓	CR ↓
Full	32.554	0.9488	0.1282	21.21	8.32%
w/o Reg.	32.535	0.9486	0.1277	22.46	8.91%
w/o Prg.	32.448	0.9480	0.1285	20.37	7.97%
w/o Adp.	32.412	0.9476	0.1291	19.58	8.13%
w/o Rfn.	31.205	0.9398	0.1353	20.62	8.86%

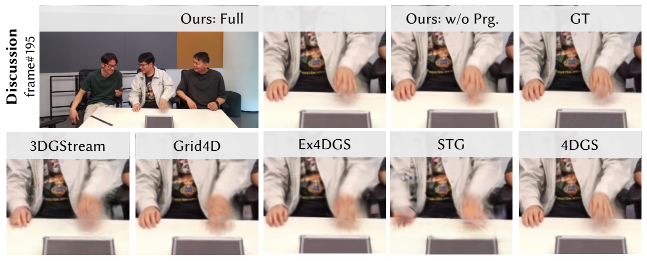


Fig. 9. **Comparisons and ablation on progressive frame sampling.** Our full method reconstructs the fast-moving hand in better quality.

We also evaluate the scalability of the proposed 4DGV method. We first compare our method with the TGH [Xu et al. 2024c] method (which is most recently proposed for handling long videos) on long video sequences. Table 3 reports the results. Our method outperform other methods in rendering quality except for the *Flame Salmon* scene from the Neu3DV dataset. On average, our compact representation is 36% smaller than TGH. Next, we report the results of our 4DGV approach for reconstructing 6000-frame videos within our captured DeskGames dataset in Table 5.

5.4 Ablations

We conduct ablation studies based on the Neu3DV [Li et al. 2022b] dataset, to verify the effectiveness of each module and to explain the configuration choices in our 4DGV approach. We first ablate four key components from our method in Table 6:

(1) **w/o Reg.**: We remove the physics-inspired ARAP and temporal smoothness regularizations during deformation, resulting in slightly decreased rendering quality and increased model size.
(2) **w/o Prg.**: We adopt uniform frame sampling within each group instead of progressive frame sampling from near to far during deformation training. This ablation causes non-negligible drops in rendering quality as it cannot handle fast and large motions (see Figure 9 for comparison).
(3) **w/o Adp.**: We remove the adaptive static-point conversion, which degrades the rendering quality due to the lack of layering flexibility (see Figure 4 for example).
(4) **w/o Rfn.**: We remove the re-initialization refinement at keyframes between deformation groups, which tends to accumulate errors from previous reconstruction, as shown in Figure 10.

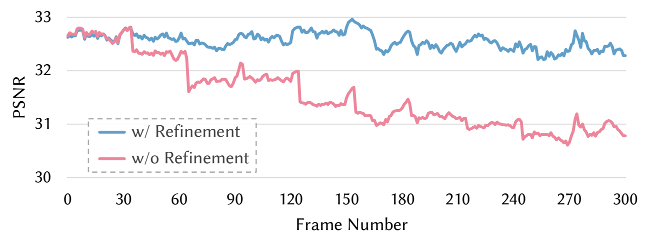


Fig. 10. **Ablation on refinement initialization.** Per-frame metrics on the Neu3DV dataset reveal the error accumulation issue without refinement. The refinement initialization on keyframe timestamps stops the error propagation.

Table 7. **Ablation on group length**. We compare the PSNR, model size (in MB), training time (in minutes) when modeling different number of frames within a deformation group.

Group Length	10	20	**30**	40	50
PSNR ↑	32.583	32.574	32.554	32.182	32.207
Size ↓	36.74	26.24	21.21	19.22	16.44
Time ↓	102.8	88.8	72.5	65.5	64.0

The comparisons in Table 6 generally verify the effectiveness of our method and demonstrate that our method achieves the ideal balance between rendering results and model sizes.

We then investigate how different frame group lengths affect our method. We report the results of our method trained with different frame group lengths in Table 7. We can see that larger group length intuitively results in lower rendering quality and smaller model sizes, as fewer groups of deformable Gaussians would be used. The increased training time when using smaller group lengths is caused by more frequent re-initialization and motion layering on keyframe timestamps. We set the frame group length to 30 for a balance between the rendering quality, model size, and training time.

Our compressed 4DGV representation supports real-time decoding and playback over the Internet due to fast decoding and the reduced model size. In our experiment, the online decoding of the deformation offsets takes an average of 3.9 ms per frame on the Neu3DV dataset. We now investigate the effectiveness of motion layering, Gaussian point quantization, and deformation encoding in reducing the model size in Table 8:

(1) **Config.#1**: Directly chaining independent GOGs leads to significant redundancy in both storage and computation.
(2) **Config.#2**: Motion layering reduces the total point size by 5× by reusing static points across groups. It also reduces the number of dynamic points to be modeled in each group, leading to training acceleration and reduction in deformation data. Meanwhile, we observe improved rendering quality with motion layering. The reasons are two-fold: First, allowing all points to deform over time would deteriorate the quality in those completely static regions. Second, fewer dynamic points would reduce the collisions in our 4D-hashgrid-based representation.

Table 8. **Ablation on model size.** We ablate different modules to evaluate their impact on the reconstructed model size on the Neu3DV dataset. We report the model size (in MB), PSNR, and reconstruction time (in minutes). *: as the extracted offsets are larger than the deformation fields, we report the size of the deformation fields instead.

Config.	#1	#2	#3	**Full**
Motion Layering		✓	✓	✓
Point Quantization			✓	✓
Deformation Encoding				✓
Size (points)	663	120	17.2	12.8
Size (deformations)	479*	220	238	8.4
Size (whole model)	1142	340	255	21.2
Compression Rate	100%	30%	22%	1.9%
PSNR	32.28	32.56	32.56	32.55
Time	194.5	62.7	72.5	73.4

Table 9. **Ablation on Gaussian point quantization.** Columns 2-5 list the bit size used for bit quantization, and the sixth column lists the codebook size for vector quantization. "-" denotes no quantization. We report the PSNR, the model size (in MB), and the compression rate on the Neu3DV-*cut_roasted_beef* frame#0. Our default configuration is highlighted.

Config.	$\mathbf{q}$	o	$\mathbf{s}$	$\mathbf{c}^{base}$	$\mathbf{c}^{rest}$	PSNR ↑	Size ↓	CR ↓
#1	-	-	-	-	-	34.26	37.14	100%
#2	-	-	-	-	2^{10}	34.26	10.31	28%
#3	8	8	12	8	2^{10}	34.21	3.65	9.8%
#4	8	8	8	8	2^{10}	32.01	2.97	8.0%
#5	8	8	12	8	2^{8}	33.91	3.55	9.6%
#6	8	8	12	8	No SH	33.88	3.50	9.4%

(3) **Config.#3**: The Gaussian point quantization can compress the point cloud size by 7×, at the cost of slightly increased training time.
(4) **Full**: The point deformation offsets extracted from the trained deformation fields are efficiently compressed by 28× using the H.265 codec. Our full strategy acheives a total compression rate of 1.85% compared to config.#1.

We then compare different parameter choices on Gaussian point quantization in Table 9. Quantization for $\mathbf{c}^{rest}$ leads to large compression rate at the cost of the rendering quality drop in highly view-dependent area (Fig. 11 top). Quantization for $\mathbf{q}, o, \mathbf{c}^{base}$ using 8 bits further reduce the model size with negligible quality loss. Our default setting uses 12 bits to quantize the point scaling $\mathbf{s}$. In contrast, using only 8 bits leads to notable degradation in rendering quality. We can also see that removing or reducing the codebook size for the high-order SH coefficients $\mathbf{c}^{rest}$ would degrade the rendering quality.

Finally, we evaluate the different compression configurations of different source data and bitrate in deformation offset encoding. We also test using DRACO [Google 2017] to further compress the Gaussian point cloud. The results are reported in Table 10. We involves

Table 10. **Ablation on deformation encoding.** We report the PSNR, model size (in MB) after compression, and compression rate (CR) on the Neu3DV dataset. "-" denotes no compression. Our default configuration is highlighted.

Config.	Source	CRF_{265}	QP_{DRC}	PSNR ↑	Size ↓	CR ↓
#1	-	-	-	32.561	254.9	100%
#2	*ofs*	20	-	32.554	23.13	9.1%
#3	*ofs*	20	20	32.554	21.21	8.3%
#4	*ofs*	20	12	32.481	20.23	7.9%
#5	*ofs*	30	12	32.439	16.65	6.5%
#6	*ori*	20	20	32.347	17.38	6.8%
#7	*dif*	20	20	32.113	23.38	9.2%

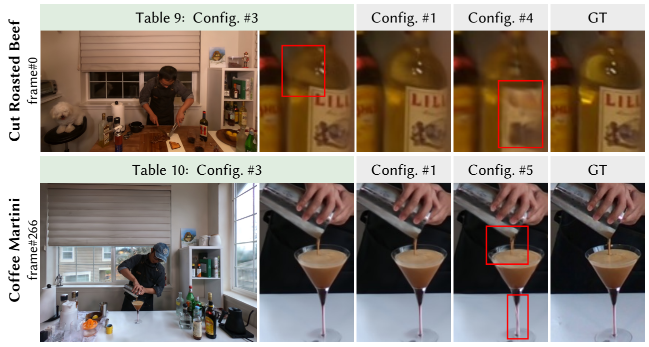


Fig. 11. **Qualitative comparisons on the GOG compression.** Top: comparisons of Gaussian point quantization configurations in Table 9. Bottom: comparisons of deformation encoding configurations in Table 10.

three different source data for H.265 encoding: "*ofs*" directly encodes the offset $\{\Delta\mathbf{x}, \Delta\mathbf{q}\}$, "*ori*" encodes the original value by adding the attributes of canonical points with the offsets $\{\mathbf{x}+\Delta\mathbf{x}, \mathbf{q}+\Delta\mathbf{q}\}$, and "*dif*" encodes the differences between offsets of adjacent timestamps $\{\Delta\mathbf{x}_{t+1}-\Delta\mathbf{x}_t, \Delta\mathbf{q}_{t+1}-\Delta\mathbf{q}_t\}$. The Constant Rate Factor in H.265 [Sullivan et al. 2012; Sze et al. 2014] codec (CRF_{265}) controls the balance between video quality and file size (lower CRF values result in higher video quality with larger file sizes). The QP_{DRC} denotes the Quantization bit used for point Position in DRACO. We choose the Config.#3 as default, which leads to minimal performance drop (Fig. 11 bottom) in rendering quality while reducing the model size by 10×.

6 CONCLUSION

In this paper, we have proposed a novel 4D Gaussian Video (4DGV) approach for creating and streaming high-fidelity volumetric videos for dynamic scenes. Our method learns a novel streamable group of Gaussians (GOG) representation based on motion layering, where each GOG obtains static and dynamic points via lifting 2D segmentation into 3D in motion layering and represents the deformation of dynamic points as the temporal offsets of their attributes. We propose 3D lifting and adaptive static-point conversion to handle the appearance change (*e.g.*, moving shadows and reflections) of static objects through optimization, and the progressive frame-sampling

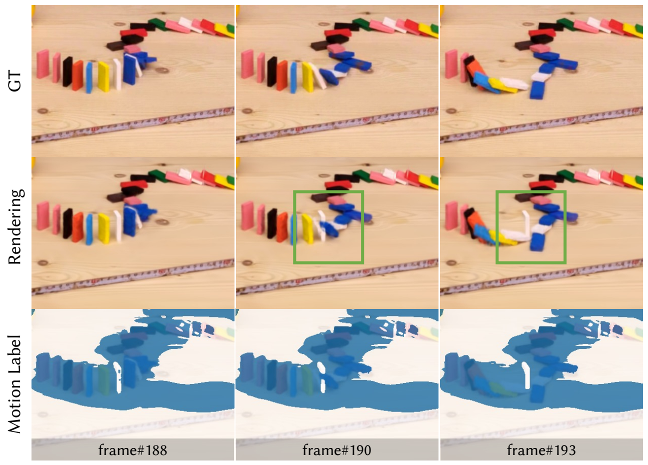


Fig. 12. **A failure case on DeskGames-*Dominos*.** When the small dominoes fall quickly, we observe both motion layering (blue: dynamic, white: static) and deformation rendering errors.

strategy to facilitate optimization of offsets for the dynamic points. In addition, we investigate compression techniques to support real-time streaming of 4DGV and show that our GOG representation can be significantly compressed without sacrificing accuracy. We have conducted extensive experiments on standard benchmarks, which demonstrate that our method outperforms state-of-the-art volumetric video approaches, producing superior rendering quality for long videos with complex motions while requiring minimum storage overheads.

For limitations, first, the adaptive motion layering may still fail to label small and fast-moving objects (Fig. 12). Second, our method lacks long-term correspondence for dynamic points since we model the scene motion within each group separately. This may cause our method to lose track of reappearing objects.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. We also thank Tongchen Zhang, Yaohui Niu, and Muwei Zhou for their assistance with dataset capturing, as well as Xiuchao Wu and Xin Zhang for their insightful discussions. Weiwei Xu is partially supported by NSFC grant No. 62421003, and Xinguo Liu is partially supported by NSFC grant No. 62032011. This paper is partially supported by the Yongjiang Innovation Project No. 2025Z062, a project from the Hong Kong Productivity Council (Ref.: 9231463), Ant Group, and Information Technology Center and State Key Lab of CAD&CG, Zhejiang University.

REFERENCES

Benjamin Attal, Jia-Bin Huang, Christian Richardt, Michael Zollhöfer, Johannes Kopf, Matthew O'Toole, and Changil Kim. 2023. HyperReel: High-Fidelity 6-DoF Video With Ray-Conditioned Sampling. In *CVPR*.
Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. 2023. Zoedepth: Zero-shot transfer by combining relative and metric depth. *arXiv:2302.12288* (2023).
Thomas Brox and Jitendra Malik. 2010. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE TPAMI* (2010).

Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew DuVall, Jason Dourgarian, Jay Busch, Matt Whalen, and Paul Debevec. 2020. Immersive Light Field Video with a Layered Mesh Representation. *ACM TOG* (2020).

Ang Cao and Justin Johnson. 2023. HexPlane: A Fast Representation for Dynamic Scenes. In *CVPR*.

Jason Chang, Donglai Wei, and John W Fisher. 2013. A video representation using temporal superpixels. In *CVPR*.

Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. 2024. HAC: Hash-grid Assisted Context for 3D Gaussian Splatting Compression. In *ECCV*.

Carl Doersch, Ankush Gupta, Larisa Markeeva, Adria Recasens, Lucas Smaira, Yusuf Aytar, Joao Carreira, Andrew Zisserman, and Yi Yang. 2022. Tap-vid: A benchmark for tracking any point in a video. In *NeurIPS*.

Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. 2023. Tapir: Tracking any point with per-frame initialization and temporal refinement. In *ICCV*.

Zheng Dong, Ke Xu, Yaoan Gao, Qilin Sun, Hujun Bao, Weiwei Xu, and Rynson WH Lau. 2023. SAILOR: Synergizing Radiance and Occupancy Fields for Live Human Performance Capture. *ACM TOG* (2023).

Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 2024. 4D-Rotor Gaussian Splatting: Towards Efficient Novel-View Synthesis for Dynamic Scenes. In *SIGGRAPH*.

Jarek Duda. 2014. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. *arXiv:1311.2540* (2014).

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2023. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. *arXiv:2311.17245* (2023).

Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. 2023. K-Planes: Explicit Radiance Fields in Space, Time, and Appearance. In *CVPR*.

Bastian Goldlücke, Marcus A Magnor, and Bennett Wilburn. 2002. Hardware-Accelerated Dynamic Light Field Rendering.. In *VMV*.

Google. 2017. *Draco 3D Graphics Compression.* https://github.com/google/draco

Zeqi Gu, Wenqi Xian, Noah Snavely, and Abe Davis. 2023. Factormatte: Redefining video matting for re-composition tasks. *ACM TOG* (2023).

Adam W Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. 2022. Particle video revisited: Tracking through occlusions using point trajectories. In *ECCV*.

Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. 2024. SC-GS: Sparse-Controlled Gaussian Splatting for Editable Dynamic Scenes. In *CVPR*.

Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. 2018. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *CVPR*.

ISO. 2018a. *PCC WD G-PCC (Geometry-Based PCC)*. Standard. International Organization for Standardization.

ISO. 2018b. *PCC WD V-PCC (Video-Based PCC)*. Standard. International Organization for Standardization.

R. Jain and K. Wakimoto. 1995. Multiple perspective interactive video. In *International Conference on Multimedia Computing and Systems*.

Joel Janai, Fatma Guney, Anurag Ranjan, Michael Black, and Andreas Geiger. 2018. Unsupervised learning of multi-frame optical flow with occlusions. In *ECCV*.

H. Kalva. 2006. The H.264 Video Coding Standard. *IEEE MultiMedia* (2006).

Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. 2024. Cotracker: It is better to track together. In *ECCV*.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM TOG* (2023).

Kevin Kwok. 2023. *Splat: A WebGL Implementation of A Real-time Renderer for 3D Gaussian Splatting for Real-Time Radiance Field Rendering.* https://github.com/antimatter15/splat

Junoh Lee, ChangYeon Won, Hyunjun Jung, Inhwan Bae, and Hae-Gon Jeon. 2024c. Fully Explicit Dynamic Guassian Splatting. In *NeurIPS*.

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. 2024b. Compact 3D Gaussian Representation for Radiance Field. In *CVPR*.

Yao-Chih Lee, Erika Lu, Sarah Rumbley, Michal Geyer, Jia-Bin Huang, Tali Dekel, and Forrester Cole. 2024a. Generative Omnimatte: Learning to Decompose Video into Layers. *arXiv:2411.16683* (2024).

Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Ping Tan. 2022a. Streaming radiance fields for 3d video synthesis. In *NeurIPS*.

Tianye Li, Mira Slavcheva, Michael Zollhöfer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. 2022b. Neural 3D Video Synthesis From Multi-View Video. In *CVPR*.

Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. 2024. Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis. In *CVPR*.

Geng Lin, Chen Gao, Jia-Bin Huang, Changil Kim, Yipeng Wang, Matthias Zwicker, and Ayush Saraf. 2023. OmnimatteRF: Robust Omnimatte with 3D Background Modeling. In *ICCV*.

Haotong Lin, Sida Peng, Zhen Xu, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. 2022. Efficient Neural Radiance Fields for Interactive Free-viewpoint Video. In *SIGGRAPH Asia Conference Proceedings*.

Han Ling, Quansen Sun, Zhenwen Ren, Yazhou Liu, Hongyuan Wang, and Zichen Wang. 2022. Scale-flow: Estimating 3d motion from video. In *ACM MM*.

Erika Lu, Forrester Cole, Tali Dekel, Andrew Zisserman, William T Freeman, and Michael Rubinstein. 2021. Omnimatte: Associating objects and their effects in video. In *CVPR*.

Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. 2024. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *CVPR*.

Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. 2024. Taming 3DGS: High-Quality Radiance Fields with Limited Resources. In *SIGGRAPH Asia*.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* (2021).

G.M. Morton. 1966. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing.* International Business Machines Company. https://books.google.com/books?id=9FFdHAAACAAJ

KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. 2024. CompGS: Smaller and Faster Gaussian Splatting with Vector Quantization. In *ECCV*.

Richard A Newcombe, Dieter Fox, and Steven M Seitz. 2015. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *CVPR*.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. 2024. Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis. In *CVPR*.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. 2023. Dinov2: Learning robust visual features without supervision. *arXiv:2304.07193* (2023).

Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. 2021. HyperNeRF: a higher-dimensional representation for topologically varying neural radiance fields. *ACM TOG* (2021).

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

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. 2024a. Sam 2: Segment anything in images and videos. *arXiv:2408.00714* (2024).

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. 2024b. SAM 2: Segment Anything in Images and Videos. *arXiv:2408.00714* (2024).

Zhile Ren, Orazio Gallo, Deqing Sun, Ming-Hsuan Yang, Erik B Sudderth, and Jan Kautz. 2019. A fusion approach for multi-frame optical flow estimation. In *WACV*.

Michael Rubinstein, Ce Liu, and William T Freeman. 2012. Towards longer long-range motion trajectories. In *BMVC*.

Neus Sabater, Guillaume Boisson, Benoit Vandame, Paul Kerbiriou, Frederic Babon, Matthieu Hog, Remy Gendrot, Tristan Langlois, Olivier Bureller, Arno Schubert, and Valerie Allie. 2017. Dataset and Pipeline for Multi-View Light-Field Video. In *CVPR Workshops*.

Peter Sand and Seth Teller. 2008. Particle video: Long-range motion estimation using point trajectories. *IJCV* (2008).

Hartmut Schirmacher, Li Ming, and Hans-Peter Seidel. 2001. On-the-Fly Processing of Generalized Lumigraphs. *Computer Graphics Forum* (2001).

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *CVPR*.

Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. 2023. NeRFPlayer: A Streamable Dynamic Scene Representation with Decomposed Neural Radiance Fields. *IEEE TVCG* (2023).

Yunzhou Song, Jiahui Lei, Ziyun Wang, Lingjie Liu, and Kostas Daniilidis. 2024. Track everything everywhere fast and robustly. In *ECCV*.

Mohammed Suhail, Erika Lu, Zhengqi Li, Noah Snavely, Leonid Sigal, and Forrester Cole. 2023. Omnimatte3D: Associating Objects and Their Effects in Unconstrained Monocular Video. In *CVPR*.

Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE TCSVT* (2012).

Deqing Sun, Erik Sudderth, and Michael Black. 2010. Layered image motion with explicit occlusions, temporal consistency, and depth ordering. In *NeurIPS*.

Jiakai Sun, Han Jiao, Guangyuan Li, Zhanjie Zhang, Lei Zhao, and Wei Xing. 2024. 3DGStream: On-the-Fly Training of 3D Gaussians for Efficient Streaming of Photo-Realistic Free-Viewpoint Videos. In *CVPR*.

Vivienne Sze, Madhukar Budagavi, and Gary J. Sullivan. 2014. *High Efficiency Video Coding (HEVC): Algorithms and Architectures.* Springer Publishing Company, Incorporated.

Zachary Teed and Jia Deng. 2020. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*.
Yi Tian and Juan Andrade-Cetto. 2024. SDformerFlow: Spatiotemporal swin spikeformer for event-based optical flow estimation. *arXiv:2409.04082* (2024).
Sundar Vedula, Simon Baker, Steven Seitz, and Takeo Kanade. 2000. Shape and motion carving in 6D. In *CVPR*.
Carl Vondrick, Abhinav Shrivastava, Alireza Fathi, Sergio Guadarrama, and Kevin Murphy. 2018. Tracking emerges by colorizing videos. In *ECCV*.
Feng Wang, Zilong Chen, Guokang Wang, Yafei Song, and Huaping Liu. 2023b. Masked space-time hash encoding for efficient dynamic scene reconstruction. *Advances in neural information processing systems* 36 (2023), 70497–70510.
Feng Wang, Sinan Tan, Xinghang Li, Zeyue Tian, Yafei Song, and Huaping Liu. 2023d. Mixed Neural Voxels for Fast Multi-view Video Synthesis. In *ICCV*.
Henan Wang, Hanxin Zhu, Tianyu He, Runsen Feng, Jiajun Deng, Jiang Bian, and Zhibo Chen. 2024d. End-to-End Rate-Distortion Optimized 3D Gaussian Representation. In *ECCV*.
Liao Wang, Kaixin Yao, Chengcheng Guo, Zhirui Zhang, Qiang Hu, Jingyi Yu, Lan Xu, and Minye Wu. 2024b. VideoRF: Rendering Dynamic Radiance Fields as 2D Feature Video Streams. In *CVPR*.
Penghao Wang, Zhirui Zhang, Liao Wang, Kaixin Yao, Siyuan Xie, Jingyi Yu, Minye Wu, and Lan Xu. 2024c. V3: Viewing Volumetric Videos on Mobiles via Streamable 2D Dynamic Gaussians. *ACM TOG* (2024).
Qianqian Wang, Yen-Yu Chang, Ruojin Cai, Zhengqi Li, Bharath Hariharan, Aleksander Holynski, and Noah Snavely. 2023a. Tracking everything everywhere all at once. In *ICCV*.
Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. 2023c. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *ICCV*.
Yihan Wang, Lahav Lipson, and Jia Deng. 2024a. SEA-RAFT: Simple, Efficient, Accurate RAFT fornbsp;Optical Flow. In *ECCV*.
Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. 2013. DeepFlow: Large displacement optical flow with deep matching. In *ICCV*.
Chung-Yi Weng, Brian Curless, Pratul P Srinivasan, Jonathan T Barron, and Ira Kemelmacher-Shlizerman. 2022. Humannerf: Free-viewpoint rendering of moving people from monocular video. In *CVPR*.
T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. 2003. Overview of the H.264/AVC video coding standard. *IEEE TCSVT* (2003).
Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 2024a. 4D Gaussian Splatting for Real-Time Dynamic Scene Rendering. In *CVPR*.
Tong Wu, Yu-Jie Yuan, Ling-Xiao Zhang, Jie Yang, Yan-Pei Cao, Ling-Qi Yan, and Lin Gao. 2024b. Recent advances in 3D Gaussian splatting. *Comput. Vis. Media* 10, 4 (August 2024), 613–642. https://doi.org/10.1007/s41095-024-0436-y
Jiawei Xu, Zexin Fan, Jian Yang, and Jin Xie. 2024a. Grid4D: 4D Decomposed Hash Encoding for High-fidelity Dynamic Scene Rendering. In *NeurIPS*.
Zhen Xu, Sida Peng, Haotong Lin, Guangzhao He, Jiaming Sun, Yujun Shen, Hujun Bao, and Xiaowei Zhou. 2024b. 4K4D: Real-Time 4D View Synthesis at 4K Resolution. In *CVPR*.
Zhen Xu, Yinghao Xu, Zhiyuan Yu, Sida Peng, Jiaming Sun, Hujun Bao, and Xiaowei Zhou. 2024c. Representing Long Volumetric Video with Temporal Gaussian Hierarchy. *ACM TOG* (2024).
Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. 2024a. Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction. In *CVPR*.
Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. 2024b. Real-time Photorealistic Dynamic Scene Representation and Rendering with 4D Gaussian Splatting. In *ICLR*.
Ye Zhang and Chandra Kambhamettu. 2001. On 3D scene flow and structure estimation. In *CVPR*.
Ruijie Zhu, Yanzhe Liang, Hanzhi Chang, Jiacheng Deng, Jiahao Lu, Wenfei Yang, Tianzhu Zhang, and Yongdong Zhang. 2024. Motiongs: Exploring explicit motion guidance for deformable 3d gaussian splatting. *arXiv:2410.07707* (2024).
C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. 2004. High-quality video view interpolation using a layered representation. *ACM TOG* (2004).

A HYPERPARAMETERS

We set the number of nearest neighbors cluster to 8, and the local time window in L_{temp} to 2 frames. Other hyperparameters settings are listed in Table 11. All default hyperparameters are used for the Neu3DV [Li et al. 2022b] dataset. We empirically set CRF_{265} and QP_{DRC} to 14 on Technicolor [Sabater et al. 2017] dataset, and use the group length of 20 on the MeetRoom [Li et al. 2022a] dataset.

Table 11. Default hyperparameter values.

λ	λ_{arap}	λ_{temp}	λ_{raft}	λ_{seg}^{dyn}	λ_{seg}^{nn}	λ_{adp}^{grad}	λ_{adp}^{nn}
0.2	0.01	0.01	2	0.3	0.5	$3e^{-4}$	0.6

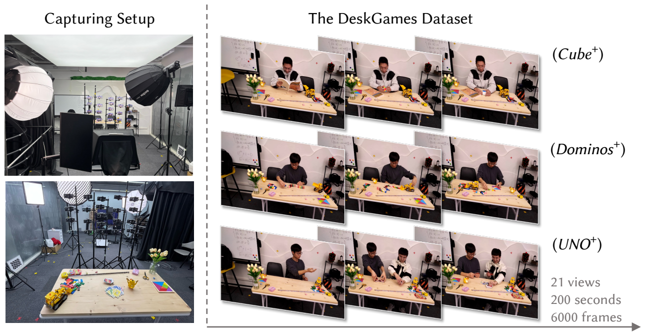


Fig. 13. Our capture setup (left) and image samples (right).

B DATASET DETAILS

We evaluate the novel-view rendering quality of our 4D Gaussian Video (4DGV) method against existing methods on four datasets, including three public datasets (i.e., Neu3DV [Li et al. 2022b], MeetRoom [Li et al. 2022a], and Technicolor [Sabater et al. 2017]), as well as our DeskGames dataset.

The Neu3DV dataset [Li et al. 2022b] contains six indoor cooking scenes, each captured from 18 to 21 viewpoints in 300 frames, including challenging effects such as flames, smoke, and fluids. We use the downsampled images at a resolution of 1, 352×1, 014 with camera index 1-17 for training, and the image with camera index 0 for testing, following the established convention [Li et al. 2022b; Yang et al. 2024b].

The MeetRoom dataset [Li et al. 2022a] records four indoor scenes with human activities, including walking, trimming, wearing VR devices, and discussion. This dataset consists of 300 frames of 13-viewpoints, at a resolution of 1, 280×720, where the images with camera index 0 are reserved for testing [Li et al. 2022a; Sun et al. 2024].

The Technicolor dataset [Sabater et al. 2017] features scenes with more complex backgrounds and motions. Following prior works [Attal et al. 2023; Lee et al. 2024c; Li et al. 2024], we conduct evaluation on the five scenes (i.e., *Birthday*, *Fabien*, *Painter*, *Theater*, and *Train*) at the full resolution of 2, 048×1, 088, using the images with camera index 10 for testing. Unlike previous works that only selected 50 frames for evaluation in each scene, we utilize all available frames of these scenes, ranging from 200 to 391 frames.

The dynamic dataset we have collected, dubbed *DeskGames*, poses a more challenging benchmark for evaluation. We have built a multi-view acquisition system utilizing 21 synchronized forward-facing cellphones to capture dynamic scenes at 30 FPS, with a resolution of 2, 560×1, 440. Our synchronized capture system and the recorded three scenes are shown in Fig. 13. Following image acquistion, we employ COLMAP [Schönberger and Frahm 2016] to calculate the intrinsic and extrinsic parameters of the cameras. Our dataset comprises three long video sequences, each spanning 6,000 frames and featuring fast motions and frequent human-object interactions.

C REAL-TIME STREAMING

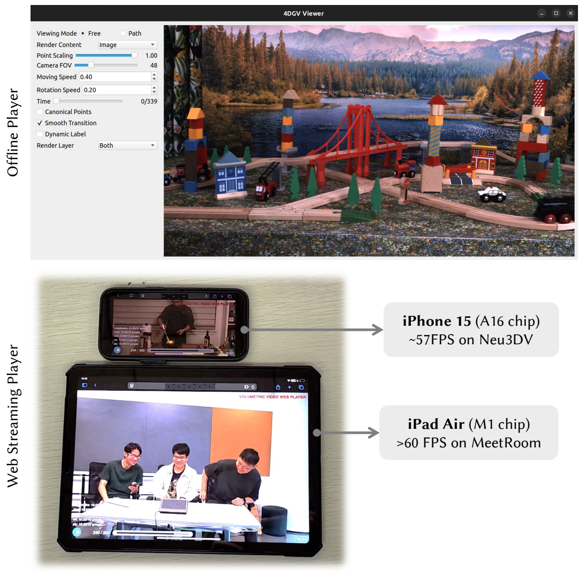
We have developed a local player for real-time playback of reconstructed 4D Gaussian videos (Fig. 14 top). Users could navigate to see the volumetric videos from any feasible viewpoint and pause the video at any time. In addition to a locally running player, we have also implemented an online web streaming player that runs in browsers by extending [Kwok 2023], enabling access to volumetric videos on almost any internet-connected device (Fig. 14 bottom). The web player supports volumetric video playback at around 60 fps, and allows users to pause videos or switch viewpoints at any time. We have successfully tested it on personal laptops (e.g., with NVIDIA GeForce RTX 3050 Laptop GPU), iPad Air, and iPhone15.

Our web player sequentially downloads data with web workers working in background threads, while the main thread handles focuses on control and rendering tasks. Upon receiving point cloud data, Gaussians are transformed into a high-resolution texture and processed in shaders. Then Gaussians are then sorted by depth on CPU to determine the rendering order. We perform Gaussian splatting using WebGL, enabling fast and smooth rendering. While fully replicating GPU-based Gaussian splatting on traditional graphics pipeline is challenging, primarily due to the absence of parallel prefix sum and per-tile depth sorting in CUDA-based 3D Gaussian splatting [Kerbl et al. 2023], the results achieved in browsers may exhibit slightly discrepancies compared to those from a CUDA-based pipeline. Nonetheless, this web player showcases the capability of our 4DGV to facilitate real-time interactive viewing of volumetric videos directly within a web browser.

D MORE EXPERIMENTAL RESULTS

Reconstruction and Compression Time Statistics. Overall, Fig. 15 shows the performance of our method on both rendering quality and storage efficiency. In Table 12, we report the average reconstruction and compression times on the Neu3DV [Li et al. 2022b] dataset, measured using a single NVIDIA RTX4090 GPU card. The H.265 [Sullivan et al. 2012; Sze et al. 2014] decoding time is evaluated in browsers. These results demonstrate the efficiency of our 4DGV in reconstructing dynamic scenes and the accessibility on the Web.

Per-scene metrics. Table 13 reports the per-scene results of our ablation studies on the Group of Gaussians (GOG) reconstruction (main paper Table 6), group length (main paper Table 7), and deformation encoding (main paper Table 10) on the Neu3DV [Li et al. 2022b] dataset. We also report the per-scene quantitative comparison results on the Neu3DV [Li et al. 2022b], MeetRoom [Li et al. 2022a], Technicolor [Sabater et al. 2017], and our DeskGames datasets, in Table 14, Table 15, Table 16, and Table 17, respectively. We compare our 4DGV method with six state-of-the-art dynamic reconstruction approaches with public available codes, including Grid4D [Xu et al. 2024a], STG [Li et al. 2024], 3DGStream [Sun et al. 2024], Ex4DGS [Lee et al. 2024c], 4DGS [Yang et al. 2024b] and V^3 [Wang et al. 2024c]. For a fair comparison with V^3 [Wang et al. 2024c], we replace the results of NeuS2 [Wang et al. 2023c] with the scene point cloud optimized by 3DGS [Kerbl et al. 2023] for key frame point initialization, to enable V^3 [Wang et al. 2024c] to handle scene-level reconstruction.



Viewing Mode [✓ Free] Path
Render Content: Image
Point Scaling: 1.00
Camera FOV: 48
Moving Speed: 0.40
Rotation Speed: 0.20
Time: 0/339
☐ Canonical Points
✓ Smooth Transition
☐ Dynamic Label
Render Layer: Both

Fig. 14. Local (top) and web streaming (bottom) players for 4DGV.

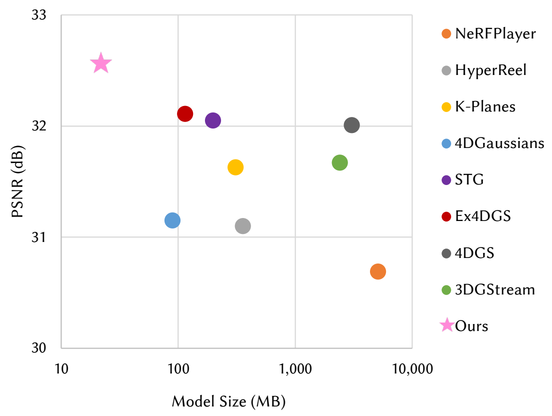


Fig. 15. Our method demonstrates state-of-the-art rendering quality and storage efficiency compared to established baselines on the Neu3DV dataset.

Table 12. **Reconstruction and compression time statistics on the Neu3DV dataset**. The initialization of the first group requires 20k iterations, while subsequent groups undergo 5k iterations for fast refinement.

Procedures	Time/scene	Time/frame	Iterations
Gaussian Point Init.	12.9 min	2.6 sec	20k(5k)/group
Motion Layering Init.	1.5 min	0.3 sec	400/group
GOG Reconstruction	47.9 min	9.6 sec	400/frame
Data I/O	10.2 min	2.0 sec	-
H.265 Encoding	0.92 min	0.18 sec	-
H.265 Decoding	117 ms	3.9 ms	-

Table 13. **Per-scene ablation metrics**. We report the PSNR and model size (in MB) on the Neu3DV dataset.

Scene	Metrics	**Default**	GOG Reconstruction: w/o				Group Length				Deformation Encoding: Config.					
			Reg.	Prog.	Adp.	Ref.	10	20	40	50	#1	#2	#4	#5	#6	#7
Coffee Martini	PSNR ↑	28.81	28.69	28.62	28.73	28.09	28.71	28.70	28.66	28.61	28.82	28.81	28.70	28.68	28.57	28.56
	Size ↓	19.05	19.44	21.84	17.58	18.67	32.96	24.01	16.97	13.26	237.6	20.98	18.18	15.68	16.18	19.88
Cook Spinach	PSNR ↑	34.01	34.08	33.95	33.90	32.14	34.16	33.99	33.79	33.83	34.01	34.01	33.95	33.92	33.85	33.51
	Size ↓	24.87	27.53	25.29	20.51	24.14	40.80	28.97	21.28	17.07	292.8	27.13	23.73	19.03	20.33	27.13
Cut Beef	PSNR ↑	33.95	34.02	33.94	33.72	32.11	34.04	34.08	33.47	33.48	33.96	33.95	33.87	33.75	33.70	33.04
	Size ↓	24.46	26.23	23.80	20.89	24.67	44.21	31.66	24.11	20.89	333.9	26.90	23.10	18.60	20.80	29.50
Flame Salmon	PSNR ↑	29.49	29.49	29.38	29.39	28.40	29.38	29.53	28.79	28.82	29.50	29.49	29.38	29.33	29.18	29.10
	Size ↓	17.26	21.78	19.82	18.85	19.70	35.29	24.04	17.07	14.70	209.0	18.85	16.55	14.55	15.05	20.25
Flame Steak	PSNR ↑	34.33	34.18	34.23	34.07	32.42	34.48	34.41	34.10	34.00	34.34	34.33	34.29	34.28	34.17	34.02
	Size ↓	23.94	22.74	17.35	21.89	20.04	39.72	27.76	22.72	18.55	261.6	25.88	22.88	17.98	18.08	24.48
Sear Steak	PSNR ↑	34.73	34.75	34.56	34.67	34.07	34.73	34.74	34.37	34.50	34.73	34.73	34.69	34.68	34.62	34.45
	Size ↓	17.69	17.04	14.11	17.01	16.51	27.46	21.98	13.18	14.17	194.3	19.06	16.96	14.06	13.86	19.06
Average	PSNR ↑	32.55	32.54	32.45	32.41	31.21	32.58	32.57	32.18	32.21	32.56	32.55	32.48	32.44	32.35	32.11
	Size ↓	21.21	22.46	20.37	19.58	20.62	36.74	26.24	19.22	16.44	254.9	23.13	20.23	16.65	17.38	23.38

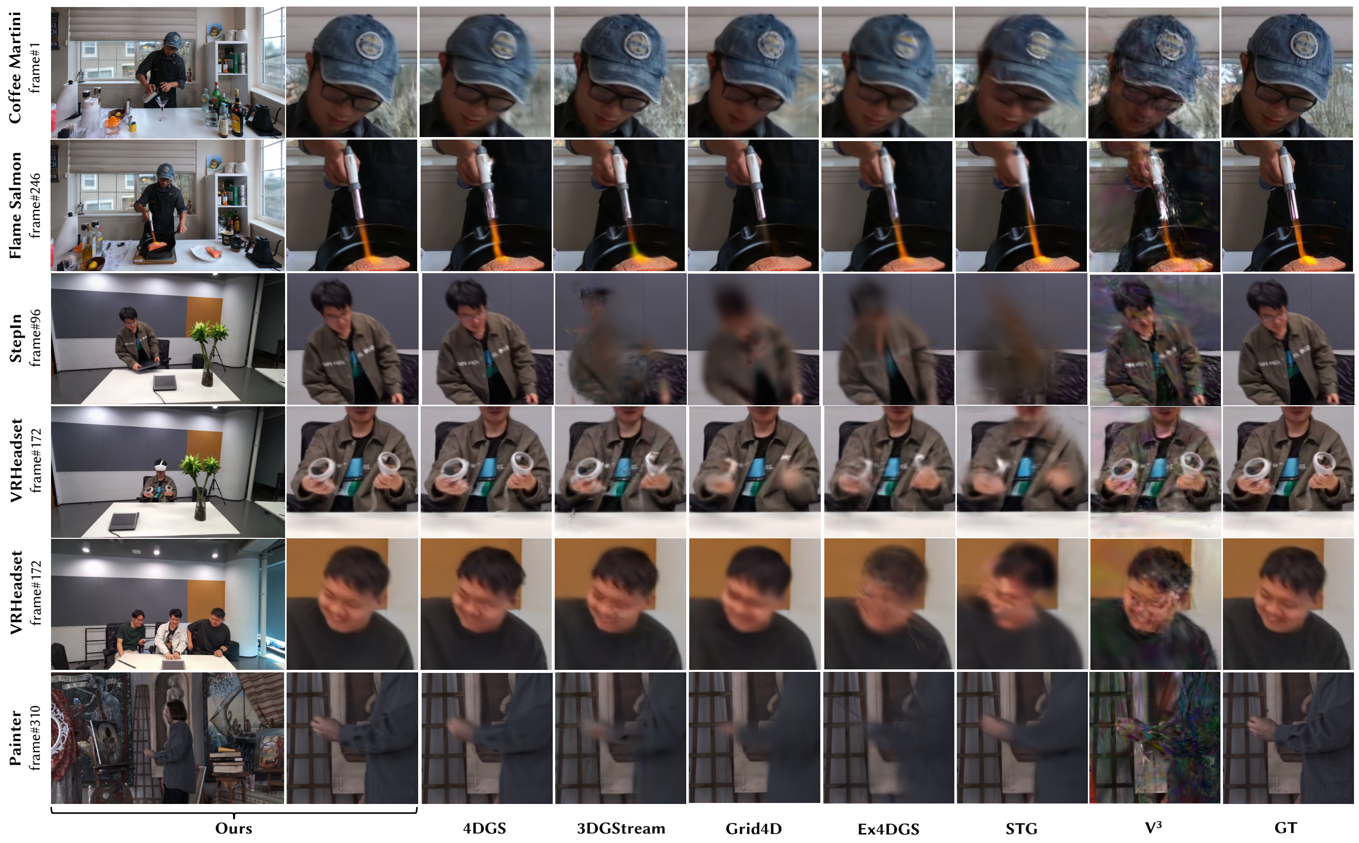


Fig. 16. More qualitative comparisons.

Table 14. **Quantitative comparisons on Neu3DV [Li et al. 2022b] dataset**. We report the PSNR, SSIM, LPIPS and model size (in MB). *: without compression.

	N3DV	PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓
Coffee Martini	Ours*	28.818	0.9197	0.1446	237.6
	Ours	28.814	0.9195	0.1447	19.05
	4DGS	27.974	0.9113	0.1544	2292
	Ex4DGS	28.051	0.9199	0.1464	133.4
	3DGStream	27.504	0.9029	0.1609	2325
	STG	28.142	0.9086	0.1712	202.0
	Grid4D	29.571	0.9146	0.1532	167.6
	V^3	23.469	0.8120	0.2854	104.2
Cook Spinach	Ours*	34.013	0.9583	0.1250	292.8
	Ours	34.008	0.9582	0.1250	24.87
	4DGS	33.230	0.9562	0.1339	2061
	Ex4DGS	32.954	0.9555	0.1301	131.8
	3DGStream	33.683	0.9553	0.1316	2304
	STG	32.824	0.9524	0.1435	222.9
	Grid4D	33.000	0.9526	0.1284	322.0
	V^3	27.179	0.8803	0.2294	123.0
Cut Beef	Ours*	33.964	0.9580	0.1305	333.9
	Ours	33.948	0.9578	0.1306	24.46
	4DGS	32.244	0.9486	0.1508	1825
	Ex4DGS	32.519	0.9588	0.1261	119.5
	3DGStream	33.779	0.9548	0.1373	2306
	STG	32.824	0.9540	0.1432	168.7
	Grid4D	32.249	0.9527	0.1263	321.4
	V^3	27.179	0.8734	0.2430	110.1
Flame Salmon	Ours*	29.500	0.9260	0.1397	209.0
	Ours	29.491	0.9255	0.1399	17.26
	4DGS	28.330	0.9141	0.1508	2765
	Ex4DGS	29.038	0.9251	0.1368	131.7
	3DGStream	28.554	0.9183	0.1444	2325
	STG	29.677	0.9258	0.1444	263.1
	Grid4D	29.949	0.9192	0.1463	168.5
	V^3	23.688	0.8115	0.2830	109.8
Flame Steak	Ours*	34.337	0.9648	0.1148	261.6
	Ours	34.332	0.9648	0.1148	23.94
	4DGS	33.164	0.9570	0.1362	1670
	Ex4DGS	33.367	0.9635	0.1135	111.1
	3DGStream	34.043	0.9643	0.1168	2301
	STG	33.572	0.9620	0.1288	233.4
	Grid4D	33.778	0.9597	0.1197	315.9
	V^3	25.831	0.8986	0.2105	116.2
Sear Steak	Ours*	34.734	0.9668	0.1143	194.3
	Ours	34.730	0.9668	0.1143	17.69
	4DGS	33.742	0.9627	0.1280	1531
	Ex4DGS	33.376	0.9624	0.1146	101.0
	3DGStream	34.458	0.9642	0.1188	2299
	STG	33.767	0.9609	0.1332	194.0
	Grid4D	34.306	0.9608	0.1191	317.9
	V^3	25.413	0.9008	0.2082	117.3

Table 15. **Quantitative comparisons on MeetRoom [Li et al. 2022a] dataset**. We report the PSNR, SSIM, LPIPS and model size (in MB). *: without compression.

	MeetRoom	PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓
Discussion	Ours*	32.648	0.9632	0.1593	310.3
	Ours	32.649	0.9631	0.1595	31.35
	4DGS	31.971	0.9569	0.1656	2486
	Ex4DGS	31.154	0.9510	0.1754	87.69
	3DGStream	30.948	0.9525	0.1695	2299
	STG	30.557	0.9432	0.1890	92.64
	Grid4D	31.091	0.9526	0.1555	301.2
	V^3	25.588	0.8611	0.3187	91.64
Step In	Ours*	30.945	0.9497	0.1815	96.88
	Ours	30.944	0.9497	0.1816	13.24
	4DGS	30.940	0.9467	0.1836	2680
	Ex4DGS	30.376	0.9379	0.1945	87.17
	3DGStream	29.275	0.9295	0.2082	2289
	STG	27.771	0.9175	0.2223	56.86
	Grid4D	28.874	0.9376	0.1763	295.8
	V^3	25.974	0.8614	0.3196	80.39
Trimming	Ours*	33.545	0.9596	0.1698	123.6
	Ours	33.545	0.9596	0.1698	16.64
	4DGS	32.747	0.9564	0.1689	1589
	Ex4DGS	32.528	0.9504	0.1779	67.45
	3DGStream	32.419	0.9537	0.1764	2289
	STG	29.697	0.9301	0.2129	62.50
	Grid4D	31.985	0.9541	0.1532	294.7
	V^3	26.776	0.8701	0.3028	74.87
VR Headset	Ours*	32.104	0.9551	0.1806	114.4
	Ours	32.106	0.9551	0.1806	15.61
	4DGS	32.091	0.9531	0.1877	2028
	Ex4DGS	30.048	0.9433	0.1943	57.52
	3DGStream	31.637	0.9490	0.1916	2290
	STG	30.034	0.9365	0.2134	73.15
	Grid4D	30.646	0.9480	0.1678	294.8
	V^3	26.135	0.8637	0.3276	80.47

Table 16. **Quantitative comparisons on Technicolor [Sabater et al. 2017] dataset**. We report the PSNR, SSIM, LPIPS and model size (in MB). *: without compression.

Technicolor		PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓
Birthday	Ours*	29.417	0.9146	0.1257	1199
	Ours	29.198	0.9095	0.0589	87.92
	4DGS	26.580	0.8291	0.2719	3306
	Ex4DGS	28.837	0.9069	0.1512	607.6
	3DGStream	22.388	0.7174	0.3981	2589
	STG	29.114	0.9170	0.1286	386.0
	Grid4D	28.890	0.9113	0.1318	440.2
	V^3	24.164	0.7637	0.2532	244.8
Fabien	Ours*	33.773	0.8656	0.3124	356.0
	Ours	32.984	0.8523	0.3262	17.68
	4DGS	32.706	0.8476	0.3695	3076
	Ex4DGS	34.318	0.8634	0.3321	112.5
	3DGStream	29.650	0.8085	0.4205	1583
	STG	34.493	0.8782	0.2795	198.1
	Grid4D	32.493	0.8487	0.3360	310.6
	V^3	28.095	0.7836	0.3789	70.65
Painter	Ours*	35.059	0.9199	0.2059	414.6
	Ours	35.004	0.9183	0.2072	34.30
	4DGS	33.030	0.8799	0.2807	2938
	Ex4DGS	33.705	0.8998	0.2469	293.6
	3DGStream	29.466	0.8270	0.3620	2956
	STG	35.795	0.9250	0.1849	678.9
	Grid4D	33.601	0.9138	0.1986	341.4
	V^3	23.260	0.6972	0.4052	180.2
Theater	Ours*	31.682	0.8701	0.2656	505.8
	Ours	31.532	0.8673	0.2686	41.13
	4DGS	28.552	0.7873	0.4058	2809
	Ex4DGS	29.742	0.8514	0.2879	283.9
	3DGStream	25.665	0.7152	0.4858	3148
	STG	30.454	0.8638	0.2513	451.6
	Grid4D	28.963	0.8569	0.2683	357.8
	V^3	25.966	0.7414	0.3806	267.9
Train	Ours*	30.119	0.9195	0.1239	125.9
	Ours	30.108	0.9193	0.1241	20.15
	4DGS	23.744	0.6455	0.4083	3220
	Ex4DGS	30.062	0.9187	0.1286	529.6
	3DGStream	20.207	0.4737	0.5643	2802
	STG	31.890	0.9385	0.0879	464.9
	Grid4D	27.878	0.9092	0.1132	440.6
	V^3	23.931	0.7418	0.2821	230.7

Table 17. **Quantitative comparisons on DeskGames dataset**. We report the PSNR, SSIM, LPIPS and model size (in MB). *: without compression.

DeskGames		PSNR ↑	SSIM ↑	LPIPS ↓	Size ↓
Cube	Ours*	30.099	0.9089	0.1828	179.6
	Ours	30.099	0.9089	0.1828	25.43
	4DGS	29.843	0.9028	0.1999	2861
	Ex4DGS	28.537	0.8926	0.1928	448.5
	3DGStream	29.971	0.9088	0.1826	2355
	STG	28.038	0.8904	0.2073	377.4
	Grid4D	22.857	0.8061	0.2965	298.7
	V^3	25.357	0.8131	0.2933	199.7
Domino	Ours*	31.463	0.9234	0.1662	408.5
	Ours	31.456	0.9233	0.1663	35.53
	4DGS	30.677	0.9134	0.1877	2805
	Ex4DGS	29.968	0.9125	0.1730	427.4
	3DGStream	30.868	0.9198	0.1754	2336
	STG	29.421	0.9171	0.1726	384.6
	Grid4D	29.427	0.9004	0.1922	346.1
	V^3	26.318	0.8350	0.2648	225.6
Reading	Ours*	31.662	0.9258	0.1672	300.0
	Ours	31.659	0.9258	0.1673	29.16
	4DGS	30.300	0.9407	0.2074	1999
	Ex4DGS	29.894	0.9098	0.1767	381.3
	3DGStream	31.448	0.9224	0.1684	2332
	STG	30.043	0.9182	0.1757	438.5
	Grid4D	23.340	0.8409	0.2450	336.4
	V^3	26.319	0.8424	0.2565	221.0
UNO	Ours*	30.373	0.9234	0.1547	591.6
	Ours	30.357	0.9232	0.1550	47.78
	4DGS	29.615	0.9020	0.2022	2798
	Ex4DGS	29.527	0.9085	0.1760	286.7
	3DGStream	29.740	0.9178	0.1635	2349
	STG	25.887	0.8813	0.2042	385.0
	Grid4D	29.547	0.9047	0.1768	354.9
	V^3	26.217	0.8281	0.2538	255.0