4K4D: Real-Time 4D View Synthesis at 4K Resolution

Zhen Xu[1] Sida Peng[1] Haotong Lin[1] Guangzhao He[1]
Jiaming Sun[2] Yujun Shen[3] Hujun Bao[1] Xiaowei Zhou[1]

[1]Zhejiang University [2]Image Derivative Inc. [3]Ant Group

Figure 1. **Photorealistic and real-time rendering of dynamic 3D scenes.** Our proposed method reconstructs a 4D neural representation from multi-view videos, which can be rendered at 1125×1536 resolution with a speed of over 200 FPS using an RTX 3090 GPU while maintaining state-of-the-art quality on the DNA-Rendering [11] dataset. It is also noteworthy that our method reaches over 80 FPS when rendering 4K images with an RTX 4090. Detailed performance under different resolutions using different GPUs can be found in Tab. 7.

Abstract

This paper targets high-fidelity and real-time view synthesis of dynamic 3D scenes at 4K resolution. Recently, some methods on dynamic view synthesis have shown impressive rendering quality. However, their speed is still limited when rendering high-resolution images. To overcome this problem, we propose 4K4D, a 4D point cloud representation that supports hardware rasterization and enables unprecedented rendering speed. Our representation is built on a 4D feature grid so that the points are naturally regularized and can be robustly optimized. In addition, we design a novel hybrid appearance model that significantly boosts the rendering quality while preserving efficiency. Moreover, we develop a differentiable depth peeling algorithm to effectively learn the proposed model from RGB videos. Experiments show that our representation can be rendered at over 400 FPS on the DNA-Rendering dataset at 1080p resolution and 80 FPS on the ENeRF-Outdoor dataset at 4K resolution using an RTX 4090 GPU, which is 30× faster than previous methods and achieves the state-of-the-art rendering quality. Our project page is available at https://zju3dv.github.io/4k4d.

1. Introduction

Dynamic view synthesis aims to reconstruct dynamic 3D scenes from captured videos and create immersive virtual playback, which is a long-standing research problem in computer vision and computer graphics. Essential to the practicality of this technique is its ability to be rendered in real time with high fidelity, enabling its application in VR/AR, sports broadcasting, and artistic performance capturing. Traditional methods [6, 12, 14, 23, 54, 55, 94] represent dynamic 3D scenes as textured mesh sequences and reconstruct them using complicated hardware. Therefore, they are typically limited to controlled environments.

Recently, implicit neural representations [17, 38, 51] have shown great success in reconstructing dynamic 3D scenes from RGB videos via differentiable rendering. For example, Li *et al*. [38] model the target scene as a dynamic radiance field and leverage volume rendering [15] to synthesize images, which are compared with input images for optimization. Despite impressive dynamic view synthesis results, existing approaches typically require seconds or even minutes to render an image at 1080p resolution due to the costly network evaluation, as discussed by Peng *et al*. [61].

Inspired by static view synthesis approaches [18, 29, 92], some dynamic view synthesis methods [2, 61, 84] increase the rendering speed by decreasing either the cost or the number of network evaluations. With these strategies, MLP Maps [61] is able to render foreground dynamic humans with a speed of 41.7 fps. However, the challenge of rendering speed still exists, since the real-time performance of MLP Maps is achieved only when synthesizing moderate-resolution images (384×512). When rendering 4K resolution images, its speed reduces to only 1.3 FPS.

In this paper, we propose a novel neural representation, named 4K4D, for modeling and rendering dynamic 3D scenes. As illustrated in Fig. 1, 4K4D significantly outperforms previous dynamic view synthesis approaches [17, 43] in terms of the rendering speed, while being competitive in the rendering quality. Our core innovation lies in a 4D point cloud representation and a hybrid appearance model. Specifically, for the dynamic scene, we obtain the coarse point cloud sequence using a space carving algorithm [33] and model the position of each point as a learnable vector. A 4D feature grid is introduced for assigning a feature vector to each point, which is fed into MLP networks to predict the point's radius, density, and spherical harmonics (SH) coefficients [52]. The 4D feature grid naturally applies spatial regularization on the point clouds and makes the optimization more robust, as supported by the results in Sec. 5.2. Based on 4K4D, we develop a differentiable depth peeling algorithm that exploits the hardware rasterizer to achieve unprecedented rendering speed.

We find that the MLP-based SH model struggles to represent the appearance of dynamic scenes. To alleviate this issue, we additionally introduce an image blending model to incorporate with the SH model to represent the scene's appearance. An important design is that we make the image blending network independent from the viewing direction, so it can be pre-computed after training to boost the rendering speed. As a two-edged sword, this strategy makes the image-blending model discrete along the viewing direction. This problem is compensated for using the continuous SH model. In contrast to 3D Gaussian Splatting [29] that uses the SH model only, our hybrid appearance model fully exploits the information captured by input images, thus effectively improving the rendering quality.

To validate the effectiveness of the proposed pipeline, we evaluate 4K4D on multiple widely used datasets for multi-view dynamic novel view synthesis, including NHR [88], ENeRF-Outdoor [43], DNA-Rendering [11], and Neural3DV [37]. Extensive experiments show that 4K4D could not only be rendered orders of magnitude faster but also notably outperform the state-of-the-art in terms of rendering quality. With an RTX 4090 GPU, our method reaches 400 FPS on the DNA-Rendering dataset at 1080p resolution and 80 FPS on the ENeRF-Outdoor dataset at 4K resolution.

2. Related Work

Neural scene representations. In the domain of novel view synthesis, various approaches have been proposed to address this challenging problem, including multi-view image-based methods [5, 7, 16, 27, 63, 97], multi-plane image representations [41, 50, 58, 78, 80, 80], light-field techniques [13, 19, 35] as well as explicit surface or voxel-based methods [12, 14, 54, 55, 94]. [12] utilizes depth sensors and multi-view stereo techniques to consolidate per-view depth information into a coherent scene geometry, producing high-quality volumetric video. These methods require intricate hardware setups and studio arrangements, thus constraining their accessibility and applicability. Recently, implicit neural scene representations [3, 21, 26, 28, 45, 46, 51, 71, 74–76, 79, 86] have attracted significant interest among researchers. NeRF [51] encodes the radiance fields of static scenes using coordinate-based Multi-Layer Perceptrons (MLP), achieving exceptional novel view synthesis quality.

Building upon NeRF, a collection of studies [24, 38, 39, 56, 57, 64, 88] has extended implicit neural representations to accommodate dynamic scenes. DyNeRF [38] introduces an additional temporal dimension to NeRF's 5D input, thereby enabling it to model temporal variations in dynamic scenes. However, NeRF-based approaches often suffer from substantial computational costs, leading to rendering times of seconds or even minutes for moderate-resolution images, which significantly limits their practicality. Another line of studies [9, 40, 85, 93] has concentrated on integrating image features into the NeRF rendering pipeline. This approach is easily applicable to dynamic scenes, as multi-view videos can be effortlessly decomposed into multi-view images. Nevertheless, the convolution operations employed in these methods, also face challenges in terms of rendering speed as the input image resolution increases, hindering the rendering efficiency of these approaches in real-world applications.

Accelerating neural scene representations. Multiple studies have focused on accelerating the rendering speed of implicit neural scene representation by distilling implicit MLP networks into explicit structures that offer fast query capabilities, including voxel grids [18, 22, 36, 53, 67, 91, 92], explicit surfaces [10, 20, 25, 32, 48, 60] and point-based representations [1, 29, 31, 34, 65, 68, 95]. These methods effectively reduce the cost or the number of NeRF's MLP evaluations required. One notable advancement is the development of 3D Gaussian Splatting (3DGS) [29] which introduces a differentiable splatting algorithm for differentiable volume rendering [4, 15]. This technique leverages semi-transparent Gaussian ellipsoids with spherical harmonics [52] to attain both high-fidelity and high-speed rendering, effectively eliminating the slow ray marching operation. However, the aforementioned acceleration techniques are only applicable to static scenes.

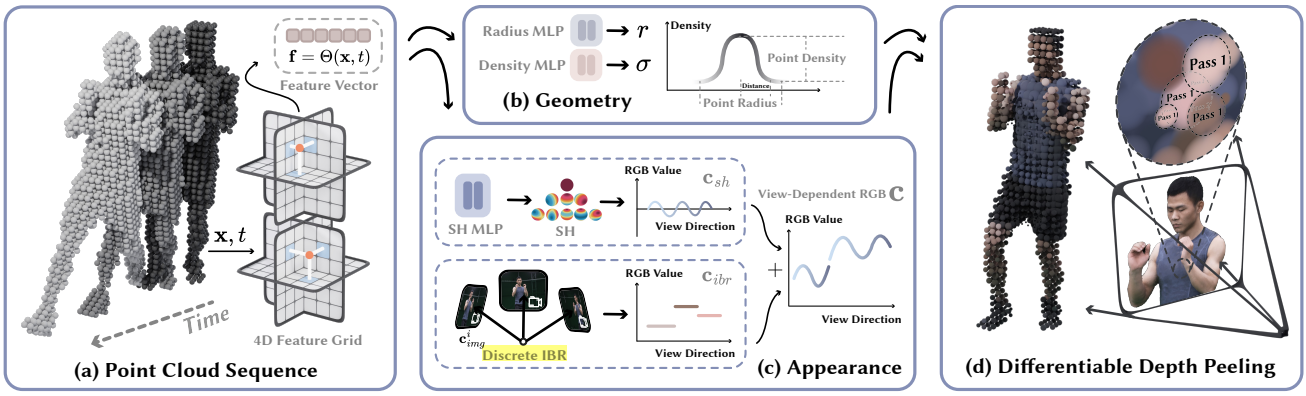

Figure 2. **Overview of our proposed pipeline.** (a) By applying the space-carving algorithm [33], we extract the initial cloud sequence $\mathbf{x}, t$ of the target scene. A 4D feature grid [17] is predefined to assign a feature vector to each point, which is then fed into MLPs for the scene geometry and appearance. (b) The geometry model is based on the point location, radius, and density, which forms a semi-transparent point cloud. (c) The appearance model consists of a piece-wise constant IBR term $\mathbf{c}_{ibr}$ and a continuous SH model $\mathbf{c}_{sh}$. (d) The proposed representation is learned from multi-view RGB videos through the differentiable depth peeling algorithm.

Inspired by the success of techniques for accelerating static neural representations, several approaches [2, 8, 42, 43, 47, 61, 70, 77, 77, 82, 83] have explored the possibility of real-time dynamic view synthesis. HyperReel [2] employs a primitive prediction module to reduce the number of network evaluations, thereby achieving real-time rendering speeds at moderate resolutions. However, it should be noted that their rendering speed decreases significantly when rendering higher-resolution images, as evidenced by experiments detailed in Sec. 5 (*e.g.*, 1.5FPS for a 1352×1014 image from the Neural3DV [38] dataset). In recent developments, a line of concurrent work [49, 87, 89, 90] has also reported real-time rendering speeds by incorporating correspondence or time-dependency into the 3DGS approach [29]. However, these methods do not show results on datasets with large and fast motions (DNA-Rendering [11] and NHR [88]) and could only achieve real-time speed at moderate resolution (800×800 [87] and 640×480 [49]). In contrast, 4K4D is capable of achieving real-time rendering even at 4K resolution while concurrently maintaining state-of-the-art view-synthesis quality on large-motion data (as elaborated in Sec. 5).

3. Proposed Approach

Given a multi-view video capturing a dynamic 3D scene, our goal is to reconstruct the target scene and perform novel view synthesis in real time. To this end, we extract coarse point clouds of the scene using the space-carving algorithm [33] (Sec. 4) and build a point cloud-based neural scene representation, which can be robustly learned from input videos and enable the hardware-accelerated rendering.

The overview of the proposed model is presented in Fig. 2. In this section, we first describe how to represent the geometry and appearance of dynamic scenes based on point clouds and neural networks (Sec. 3.1). Then, we develop a differentiable depth peeling algorithm for rendering our representation (Sec. 3.2), which is supported by the hardware rasterizer, thereby significantly improving the rendering speed. Finally, we discuss how to optimize the proposed model on input RGB videos (Sec. 3.3).

3.1. Modeling Dynamic Scenes with Point Clouds

4D embedding. Given the coarse point clouds of the target scene, we represent its dynamic geometry and appearance using neural networks and feature grids. Specifically, our method first defines six feature planes θ_{xy}, θ_{xz}, θ_{yz}, θ_{tx}, θ_{ty}, and θ_{tz}. To assign a feature vector $\mathbf{f}$ to any point $\mathbf{x}$ at frame t, we adopt the strategy of K-Planes [17] to model a 4D feature field $\Theta(\mathbf{x}, t)$ using these six planes:

$$\mathbf{f} = \Theta(\mathbf{x}, t) = \theta_{xy}(x, y) \oplus \theta_{xz}(x, z) \oplus \theta_{yz}(y, z) \oplus \theta_{tx}(t, x) \oplus \theta_{ty}(t, y) \oplus \theta_{tz}(t, z), \quad (1)$$

where $\mathbf{x} = (x, y, z)$ is the input point, and $\oplus$ indicates the concatenation operator. Please refer to K-Planes [17] for more implementation details.

Geometry model. Based on coarse point clouds, the dynamic scene geometry is represented by learning three entries on each point: position $\mathbf{p} \in R^3$, radius $r \in R$, and density $\sigma \in R$. Using these point entries, we calculate the volume density of space point $\mathbf{x}$ with respect to an image pixel $\mathbf{u}$ for the volume rendering, which will be described in Sec. 3.2. The point position $\mathbf{p}$ is modeled as an optimizable vector. The radius r and density σ are predicted by feeding the feature vector $\mathbf{f}$ in Eq. (1) to an MLP network.

Appearance model. As illustrated in Fig. 2c, we use the image blending technique and the spherical harmonics (SH)

model [52, 92] to build a hybrid appearance model, where the image blending technique represents the discrete view-dependent appearance $\mathbf{c}_{ibr}$ and the SH model represents the continuous view-dependent appearance $\mathbf{c}_{sh}$. For point $\mathbf{x}$ at frame t, its color with viewing direction $\mathbf{d}$ is:

$$\mathbf{c}(\mathbf{x}, t, \mathbf{d}) = \mathbf{c}_{ibr}(\mathbf{x}, t, \mathbf{d}) + \mathbf{c}_{sh}(\mathbf{s}, \mathbf{d}), \tag{2}$$

where $\mathbf{s}$ means SH coefficients at point $\mathbf{x}$.

The discrete view-dependent appearance $\mathbf{c}_{ibr}$ is inferred based on input images. Specifically, for a point $\mathbf{x}$, we first project it into the input image to retrieve the corresponding RGB color $\mathbf{c}^i_{img}$. Then, to blend input RGB colors, we calculate the corresponding blending weight w^i based on the point coordinate and the input image. Note that the blending weight is independent from the viewing direction. Next, to achieve the view-dependent effect, we select the N' nearest input views according to the viewing direction. Finally, the color $\mathbf{c}_{ibr}$ is computed as $\sum_{i=1}^{N'} w^i \mathbf{c}^i_{img}$. Because the N' input views are obtained through the nearest neighbor retrieval, the $\mathbf{c}_{ibr}$ is inevitably discrete along the viewing direction. To achieve the continuous view-dependent effect, we append the fine-level color $\mathbf{c}_{sh}$ represented by the SH model, as shown in Fig. 2c.

In practice, our method regresses the SH coefficients $\mathbf{s}$ by passing the point feature $\mathbf{f}$ in Eq. (1) into an MLP network. To predict the blending weight w^i in the image blending model $\mathbf{c}_{ibr}$, we first project point $\mathbf{x}$ onto the input image to retrieve the image feature $\mathbf{f}^i_{img}$, and then concatenate it with the point feature $\mathbf{f}$, which is fed into another MLP network to predict the blending weight. The image feature $\mathbf{f}^i_{img}$ is extracted using a 2D CNN network.

Discussion. Our appearance model is the key to achieving the low-storage, high-fidelity, and real-time view synthesis of dynamic scenes. There are three alternative ways to represent the dynamic appearance, but they cannot perform on par with our model. 1) Defining explicit SH coefficients on each point, as in 3D Gaussian splatting [29]. When the dimensional of SH coefficients is high and the amount of points of dynamic scenes is large, this model's size could be too big to train on a consumer GPU. 2) MLP-based SH model. Using an MLP to predict SH coefficients of each point can effectively decrease the model size. However, our experiments found that MLP-based SH model struggles to render high-quality images (Sec. 5.2). 3) Continuous view-dependent image blending model, as in ENeRF [43]. We found that representing the appearance with the image blending model has better rendering quality than with only MLP-based SH model. However, the network in ENeRF takes the viewing direction as input and thus cannot be easily pre-computed, limiting the rendering speed during inference.

In contrast to these three methods, our appearance model combines a discrete image blending model $\mathbf{c}_{ibr}$ with a continuous SH model $\mathbf{c}_{sh}$. The image blending model $\mathbf{c}_{ibr}$ boosts the rendering performance. In addition, it supports the pre-computation, as its network does not take the viewing direction as input. The SH model $\mathbf{c}_{sh}$ enables the view-dependent effect for any viewing direction. During training, our model represents the scene appearance using networks, so its model size is reasonable. During inference, we pre-compute the network outputs to achieve the real-time rendering, which will be described in Sec. 3.4.

3.2. Differentiable Depth Peeling

Our proposed dynamic scene representation can be rendered into images using the depth peeling algorithm [4]. Thanks to the point cloud representation, we are able to leverage the hardware rasterizer to significantly speed up the depth peeling process. Moreover, it is easy to make this rendering process differentiable, enabling us to learn our model from input RGB videos.

We develop a custom shader to implement the depth peeling algorithm that consists of K rendering passes. Consider a particular image pixel $\mathbf{u}$. In the first pass, our method first uses the hardware rasterizer to render point clouds onto the image, which assigns the closest-to-camera point $\mathbf{x}_0$ to the pixel $\mathbf{u}$. Denote the depth of point $\mathbf{x}_0$ as t_0. Subsequently, in the k-th rendering pass, all points with depth value t_k smaller than the recorded depth of the previous pass t_{k-1} are discarded, thereby resulting in the k-th closest-to-camera point $\mathbf{x}_k$ for the pixel $\mathbf{u}$. Discarding closer points is implemented in our custom shader, so it still supports the hardware rasterization. After K rendering passes, pixel $\mathbf{u}$ has a set of sorted points $\{\mathbf{x}_k | k = 1, ..., K\}$.

Based on the points $\{\mathbf{x}_k | k = 1, ..., K\}$, we use the volume rendering to synthesize the color of pixel $\mathbf{u}$. The densities of points $\{\mathbf{x}_k | k = 1, ..., K\}$ for pixel $\mathbf{u}$ is defined based on the distance between the projected point and pixel $\mathbf{u}$ on the 2D image:

$$\alpha(\mathbf{u}, \mathbf{x}) = \sigma \cdot \max(1 - \frac{||\pi(\mathbf{x}) - \mathbf{u}||_2^2}{r^2}, 0), \tag{3}$$

where π is the camera projection function. σ and r are the density and radius of point $\mathbf{x}$, which are described in Sec. 3.1. During training, we implement the projection function π using the PyTorch [59], so Eq. (3) is naturally differentiable. During inference, we leverage the hardware rasterization process to efficiently obtain the distance $||\pi(\mathbf{x}) - \mathbf{u}||_2^2$, which is implemented using OpenGL [72].

Denote the density of point $\mathbf{x}_k$ as α_k. The color of pixel $\mathbf{u}$ from the volume rendering is formulated as:

$$C(\mathbf{u}) = \sum_{k=1}^{K} T_k \alpha_k \mathbf{c}_k, \text{ where } T_k = \prod_{j=1}^{k-1} (1 - \alpha_j), \tag{4}$$

where $\mathbf{c}_k$ is the color of point $\mathbf{x}_k$, as described in Eq. (2).

3.3. Training

Given the rendered pixel color $C(\mathbf{u})$, we compare it with the ground-truth pixel color $C_{gt}(\mathbf{u})$ to optimize our model in an end-to-end fashion using the following loss function:

$$L_{img} = \sum_{\mathbf{u}\in\mathcal{U}} ||C(\mathbf{u}) - C_{gt}(\mathbf{u})||_2^2, \tag{5}$$

where $\mathcal{U}$ is the set of image pixels. In addition to the MSE loss L_{img}, we also apply the perceptual loss L_{lpips} [96].

$$L_{lpips} = ||\Phi(I) - \Phi(I_{gt})||_1, \tag{6}$$

where Φ is the perceptual function (a VGG16 network) and I, I_{gt} are the rendered and ground-truth images, respectively. The perceptual loss [96] computes the difference in image features extracted from the VGG model [73]. Our experiments in Sec. 5.2 show that it effectively improves the perceived quality of the rendered image.

To regularize the optimization process of our proposed representation, we additionally apply the mask supervision to dynamic regions of the target scene. We solely render point clouds of dynamic regions to obtain their masks, where the pixel value is obtained by:

$$M(\mathbf{u}) = \sum_{k=1}^{K} T_k \alpha_k, \text{ where } T_k = \prod_{j=1}^{k-1}(1-\alpha_j). \tag{7}$$

The mask loss is defined as:

$$L_{msk} = \sum_{\mathbf{u}\in\mathcal{U}'} M(\mathbf{u}) M_{gt}(\mathbf{u}), \tag{8}$$

where $\mathbf{U}'$ means the set of pixels of the rendered mask, and M_{gt} is the ground-truth mask of 2D dynamic regions. This effectively regularizes the optimization of the geometry of dynamic regions by confining it to the visual hulls.

The final loss function is defined as

$$L = L_{img} + \lambda_{lpips} L_{lpips} + \lambda_{msk} L_{msk}, \tag{9}$$

where λ_{lpips} and λ_{msk} are hyperparameters controlling weights of correspondings losses.

3.4. Inference

After training, we apply a few acceleration techniques to boost the rendering speed of our model. First, we precompute the point location $\mathbf{p}$, radius r, density σ, SH coefficients $\mathbf{s}$ and color blending weights w_i before inference, which are stored at the main memory. During rendering, these properties are asynchronously streamed onto the graphics card, overlapping rasterization with memory copy to achieve an optimal rendering speed [69, 72]. After applying this technique, the runtime computation is reduced to only a depth peeling evaluation (Sec. 3.2) and a spherical harmonics evaluation (Eq. (2)). Second, we convert the model from 32-bit floats to 16-bits for efficient memory access, which increases FPS by 20 and leads to no visible performance loss as validated in Tab. 6. Third, the number of rendering passes K for the differentiable depth peeling algorithm is reduced from 15 to 12, also leading to a 20 FPS speedup with no visual quality change. Detailed analyses of rendering speed can be found in Sec. 5.2.

4. Implementation Details

Optimization. 4K4D is trained using the PyTorch framework [59]. Using the Adam optimizer [30] with a learning rate $5e^{-3}$, our models typically converge after 800k iterations for a sequence length of 200 frames, which takes around 24 hours on a single RTX 4090 GPU. Specifically, the learning rate of point positions is set to $1e^{-5}$, and the regularization loss weights λ_{lpips} and λ_{msk} are set to $1e^{-3}$. During training, the number of passes K for the differentiable depth peeling is set to 15, and the number of nearest input views N' is set to 4. The rendering speed of our method is reported on an RTX 3090 GPU for the experiments in Sec. 5 unless otherwise stated.

Initialization of point clouds. We leverage exisiting multi-view reconstruction methods to initialize the point clouds. For dynamic regions, we use segmentation methods [44] to obtain their masks in input images and utilize the space carving algorithm [33] to extract their coarse geometry. For static background regions, we leverage foreground masks to compute the mask-weighted average of background pixels along all frames, producing background images without the foreground content. Then, an Instant-NGP [53] model is trained on these images, from which we obtain the initial point clouds. After the initialization, the number of points for the dynamic regions is typically 250k per frame, and the static background regions typically consist of 300k points.

5. Experiments

Datasets and metrics. We train and evaluate our method 4K4D on multiple widely used multi-view datasets, including DNA-Rendering [11], ENeRF-Outdoor [43], NHR [88] and Neural3DV [38]. DNA-Rendering [11] records 10-second clips of dynamic humans and objects at 15 FPS using 4K and 2K cameras with 60 views. This dataset is very challenging due to the complex clothing and fast-moving humans recorded. We conduct experiments on 4 sequences of DNA-Rendering, with 90% of the views as training set and the rest as evaluation set. ENeRF-Outdoor [43] records multiple dynamic humans and objects in an outdoor environment at 30FPS using 1080p cameras. We select three 100-frame sequences with 6 different actors (2 for each sequence) holding objects to evaluate our method 4K4D.

Figure 3. **Qualitative comparison on the ENeRF-Outdoor [43] dataset that contains 960 × 540 images.** Our method achieves much higher rendering quality and can be rendered 13× faster than ENeRF [43]. More dynamic results can be found in the supplementary video.

This dataset is difficult for dynamic view synthesis in that not only are there multiple moving humans and objects in one clip, but the background is also dynamic due to the shadow of the humans. Following Im4D [42] and NeuralBody [62], we evaluate metrics on the dynamic regions for the DNA-Rendering [11] and NHR [88] dataset, which can be obtained by predefining the 3D bounding box of the person and projecting it onto the images. For ENeRF-Outdoor [43], we jointly train the dynamic geometry and appearance of the foreground and the dynamic appearance of the background to obtain rendering results on the whole image. All images are resized with a ratio of 0.5 for evaluation and 0.375 if the original resolution is more than 2K. For DNA-Rendering, the rendered image size is 1024×1224 (and 1125×1536) and for ENeRF-Outdoor, the resolution is 960×540 during the experiments. The resolutions for Neural3DV video and NHR are 1352×1224 and 512×612 (and 384×512) respectively. Detailed dataset settings can be found in Appendix A.

Table 1. **Quantitative comparison on the DNA-Rendering [11] dataset.** Image resolutions are 1024×1224 and 1125×1536. Metrics are averaged over all scenes. Green and yellow cell colors indicate the best and the second best results, respectively.

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS
ENeRF [43]	28.108	0.972	0.056	6.011
IBRNet [85]	27.844	0.967	0.081	0.100
KPlanes [17]	27.452	0.952	0.118	0.640
Im4D [42]	28.991	0.973	0.062	15.360
Ours	31.173	0.976	0.055	203.610

5.1. Comparison Experiments

Comparison results. Qualitative and quantitative comparisons on the DNA-Rendering [11] are shown in Fig. 5 and Tab. 1 respectively. As evident in Tab. 1, our method 4K4D renders 30x faster than the SOTA real-time dynamic view synthesis method ENeRF [43] with superior rendering quality. Even when compared with concurrent work [42], our method 4K4D still achieves 13x speedup and

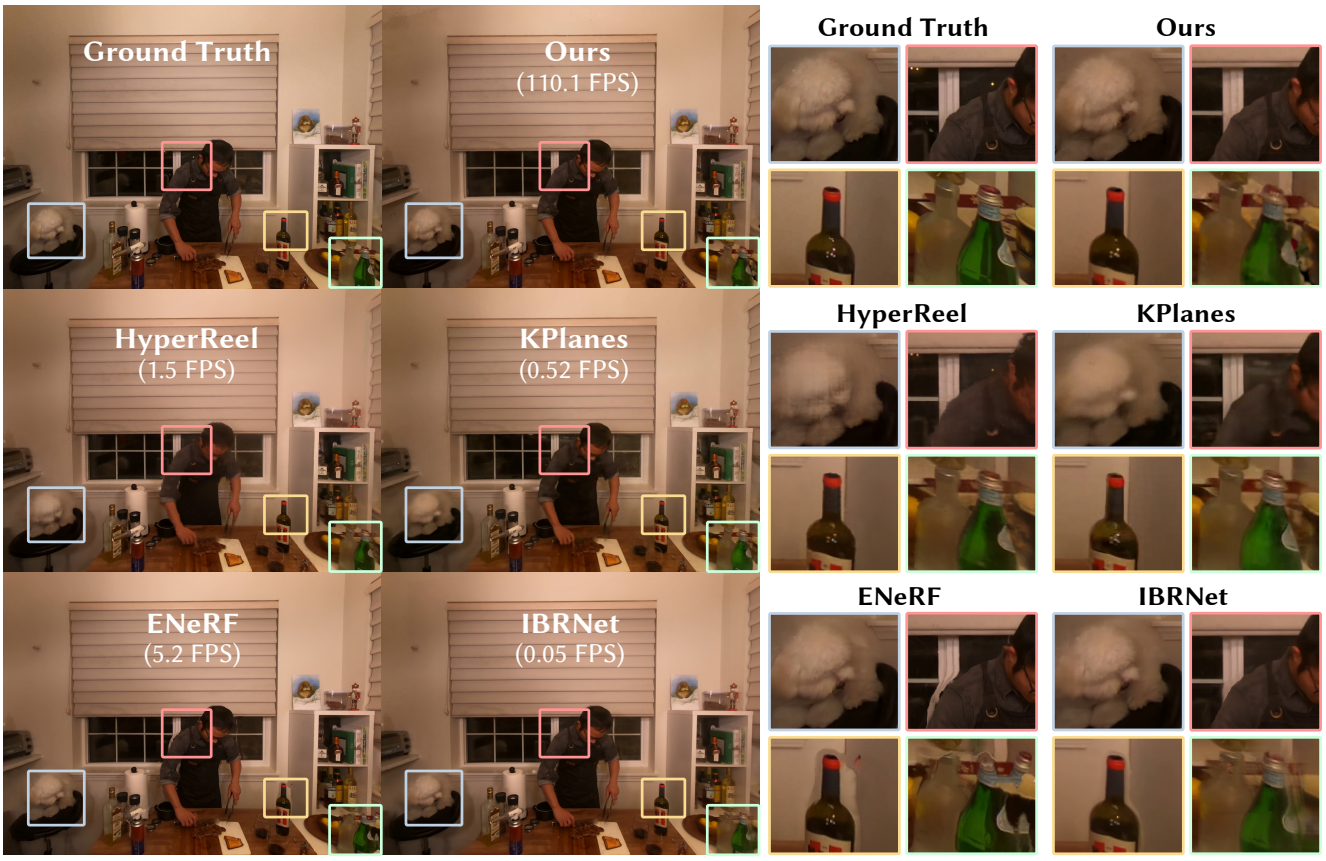

Figure 4. **Qualitative comparison on the Neural3DV [38] dataset that contains 1352×1224 images.** Our method can not only recover high-frequency details of dynamic objects but also maintain sharp edges around occlusion.

produces consistently higher quality images. As shown in Fig. 5, KPlanes [17] could not recover the highly detailed appearance and geometry of the 4D dynamic scene. Other image-based methods [42, 43, 85] produce high-quality appearance. However, they tend to produce blurry results around occlusions and edges, leading to degradation of the visual quality while maintaining interactive framerate at best. On the contrary, our method 4K4D can produce higher fidelity renderings at over 200 FPS. Fig. 3 and Tab. 2 provides qualitative and quantitaive results on the ENeRF-Outdoor [43] dataset. Even on the challenging ENeRF-Outdoor dataset with multiple actors and the background, our method 4K4D still achieves notably better results while rendering at over 140 FPS. ENeRF [43] produces blurry results on this challenging dataset, and the rendering results of IBRNet [85] contain black artifacts around the edges of the images as shown in Fig. 3. K-Planse [17] fails to reconstruct the dynamic humans and varying background regions. More comparison results on the NHR [88] dataset can be found in Appendix B.

Table 2. **Quantitative comparison on the ENeRF-Ourdoor [43] dataset.** This dataset includes 960 × 540 images. Green and yellow cell colors indicate the best and the second best results, respectively.

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS
ENeRF [43]	25.452	0.809	0.273	11.309
IBRNet [85]	24.966	0.929	0.172	0.140
KPlanes [17]	21.310	0.735	0.454	1.370
Ours	25.815	0.898	0.147	141.665

Table 3. **Quantitative comparison on the Neural3DV [38] dataset.** This dataset includes 1352×1224 images. Green and yellow cell colors indicate the best and the second best results, respectively.

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS
ENeRF [43]	30.306	0.962	0.185	5.187
IBRNet [85]	31.520	0.963	0.169	0.053
KPlanes [17]	31.610	0.961	0.182	0.518
HyperReel [2]	32.198	0.970	0.161	1.540
Ours	32.855	0.973	0.167	110.063

5.2. Ablation Studies

We perform ablation studies on the 150-frame *0013_01* sequence of the DNA-Rendering [11] dataset. Qualitative

Ground Truth **Ours** (203.6 FPS) **ENeRF** (6.0 FPS) **IBRNet** (0.1 FPS) **Im4D** (15.4 FPS) **KPlanes** (0.6 FPS)

Figure 5. **Qualitative comparison on the DNA-Rendering [11] dataset that contains 1024×1224 (and 1125×1536) images.** Our method can produce high-fidelity images at over 200 FPS while other competitors fail to produce high-quality results for highly dynamic scenes.

and quantitative results are shown in Fig. 6 and Tabs. 4 to 7.

Ablation study on the 4D embedding. The "*w/o* $\mathbf{f}$" variant removes the proposed 4D embedding (Sec. 3.1) module and replaces it with a per-frame and per-point optimizable position, radius, density, and scale. As shown in Fig. 6 and Tab. 4, the "*w/o* $\mathbf{f}$" variant produces blurry and noisy geometry without the 4D embedding Θ, which leads to the inferior rendering quality.

Ablation study on the hybrid appearance model. The "*w/o* $\mathbf{c}_{ibr}$" variant removes $\mathbf{c}_{ibr}$ in the appearance formulation Eq. (2), which not only leads to less details on the recovered appearance but also significantly impedes the quality of the geometry. Adding an additional degree for the SH coefficients does not lead to a significant performance change (PSNR 30.202 *vs.* 30.328). Comparatively, our proposed method produces high-fidelity rendering with much better details. A visualization of the view-dependent effect produced by $\mathbf{c}_{sh}$ can be found in Appendix B.

Ablation study on loss functions. As shown in Tab. 4, removing the L_{lpips} term not only reduces the perceptual quality (LPIPS score) but also leads to the degradation of other performance metrics. For the highly dynamic DNA-Rendering [11] dataset, the mask loss L_{msk} helps with regularizing the optimization of the dynamic geometry.

Table 4. **Ablation studies** on the 150-frame *0013_01* sequence of the DNA-Rendering dataset [11]. "*w/o* $\mathbf{f}$" indicates replacing the 4D embedding with a per-frame and per-point optimizable position, radius, density, and scale. See Sec. 5.2 for more detailed descriptions.

	PSNR ↑	SSIM ↑	LPIPS ↓	Model Size
w/o $\mathbf{f}$	29.779	0.967	0.057	1304.0 MiB
w/o $\mathbf{c}_{ibr}$	30.259	0.973	0.054	225.0 MiB
w/o $\mathbf{c}_{sh}$	31.946	0.981	0.040	225.0 MiB
w/o L_{lpips}	31.661	0.979	0.063	225.0 MiB
w/o L_{msk}	29.115	0.965	0.073	225.0 MiB
Ours	31.990	0.982	0.040	225.0 MiB

Storage analysis. For the 150-frame *0013_01* scene, the storage analysis of our method 4K4D is listed in Tab. 5. The point positions $\mathbf{p}$ take up the majority of the model size due to its explicit representation. The final storage cost for our method is less than 2 MB per frame with source videos included. The input images of DNA-Rendering [11] are provided in JPEG formats. We encode frames of all input images as videos using the HEVC encoder of FFmpeg with a CRF of 25 [81]. After the encoding, we obverse no change in LPIPS (0.040), no loss in SSIM (0.982), and only a 0.42% decrease in PSNR (31.990 *vs.* 31.855), which indicates that

Figure 6. **Ablation study of proposed components on the *0013_01* sequence of the DNA-Rendering dataset [11].** Removing our proposed components leads to noisy geometry and blurry appearance. Our method produces high-fidelity results with perceptually accurate shapes and colors. See Sec. 5.2 for more detailed descriptions.

Table 5. **Storage analysis of our method on the 150-frame *0013_01* sequence of the DNA-Rendering dataset [11].** "Storage" indicates the size of model stored on disk, and "Storage / Frame" indicates the per-frame size.

	Point Positions $\mathbf{p}$	4D Embedding Θ	MLPs and CNNs	Total Model Size	Encoded Video	Total Storage (*w/* Videos)
Storage	208.09 MB	16.77 MB	0.10 MB	224.96 MB	62.89 MB	287.86 MB
Storage / Frame	1.387 MB	0.112 MB	0.001 MB	1.500 MB	0.419 MB	1.919 MB

our method 4K4D is robust to the video encoding of input images. After encoding the input images as videos, the storage overhead for Image-Based Rendering (Sec. 3.1) is only 0.419 MB per frame with minimal rendering quality change.

As mentioned in Sec. 3.4, we precompute the physical properties on the point clouds for real-time rendering, which takes around 2 seconds for one frame. Although large in size (200 MB for one frame of *0013_01*), these precomputed caches only reside in the main memory and are not explicitly stored on disk, which is feasible for a modern PC. This makes our representation a form of compression, where the disk file size is small (2 MB per frame) but the information contained is very rich (200 MB per frame).

5.3. Rendering Speed Analysis

As mentioned in Sec. 3.4, we introduce a number of optimization techniques to accelerate the rendering speed of our method 4K4D, which are only made possible by our proposed hybrid geometry and appearance representation. In Tab. 6, we ablate the effectiveness and quality impact of those proposed techniques on the 150-frame *0013_01* sequence of the DNA-Rendering [11] dataset.

Table 6. **Runtime analysis of the proposed method on the *0013_01* sequence of DNA-Rendering [11].** The acceleration techniques all lead to minimal quality changes as shown by the cell coloring (green for the best and yellow for the second best). See Sec. 5.2 for detailed descriptions.

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS
w/o fp16	32.020	0.982	0.039	202.021
w/o $K = 12$	31.951	0.982	0.040	200.397
w/o Cache	31.969	0.982	0.040	22.193
w/o DDP	31.900	0.981	0.041	29.656
Ours	31.990	0.982	0.040	219.430

The effectiveness of precomputation. For real-time rendering, we precompute and cache $\mathbf{p}$, r, σ and $\mathbf{s}$ for all points and store them in the main memory. Thanks to our design choice

Table 7. **Rendering speed on different GPUs and resolutions.** The results are recorded on the first frame of the *0013_01* sequence of DNA-Rendering [11] and the *actor1_4* sequence of ENeRF-Outdoor [43] with the interactive GUI. Resolutions are set to 720p (720 × 1280), 1080p (1080 × 1920), and 4K (2160 × 3840). Even with the overhead of the interactive GUI ("***w/* GUI**"), our method still achieves unprecedented rendering speed. More real-time rendering results can be found in the supplementary video.

Dataset	Res.	RTX 3060	RTX 3090	RTX 4090
DNA-Rendering [11] ***w/* GUI**	720p	173.8 FPS	246.9 FPS	431.0 FPS
	1080p	138.7 FPS	233.1 FPS	409.8 FPS
	4K	**90.0 FPS**	**147.4 FPS**	**288.8 FPS**
ENeRF-Outdoor [43] ***w/* GUI**	720p	90.5 FPS	130.5 FPS	351.5 FPS
	1080p	66.1 FPS	103.6 FPS	249.7 FPS
	4K	**25.1 FPS**	**47.2 FPS**	**85.1 FPS**

of splitting the appearance representation into constant $\mathbf{c}_{ibr}$ and view-dependent $\mathbf{c}_{sh}$, we can also precompute and cache the per-image weights w and color $\mathbf{c}_{img}$ for all source images (Sec. 3.1). These caches take around 200MB per frame of main memory for the 150-frame 60-view scene of *0013_01* of the DNA-Rendering [11] dataset. The pre-computation enabled by our representation (Sec. 3.1) achieves a 10x speedup (Ours *vs.* "*w/o* Cache").

Differentiable depth peeling. We also make comparisons with more traditional CUDA-based differentiable point cloud rendering technique provided by PyTorch3D [66] ("*w/o* DDP") to validate the effectiveness of our proposed differentiable depth peeling algorithm (Sec. 3.2). Both our proposed DDP (Sec. 3.2) and PyTorch3D's [66] implementation use the same volume rendering equation as in Eq. (4). As shown in Tab. 6, our proposed method is more than 7 times faster than the CUDA-based one.

Other acceleration techniques. The "*w/o* fp16" variant uses the original 32-bit floating point number for computation. The "*w/o* $K = 12$" variant uses 15 passes in the depth peeling algorithm as when training. Using 16-bit floats and 12 rendering passes both lead to a 20FPS speedup.

Rendering speed on different GPUs and resolutions. We additionally report the rendering speed of our method on different hardware (RTX 3060, 3090, and 4090) with different resolutions (720p, 1080p, and 4K (2160p)) in Tab. 7. The rendering speed reported here contains the overhead of the interactive GUI ("***w/* GUI**"), making them slightly slower than those reported in Sec. 5.1. 4K4D achieves real-time rendering speed even when rendering 4K (2160p) images on commodity hardware as shown in the table.

6. Conclusion and Discussion

In this paper, we provide a neural point cloud-based representation, 4K4D, for real-time rendering of dynamic 3D scenes at 4K resolution. We build 4K4D upon a 4D feature grid to naturally regularize the points and develop a novel hybrid appearance model for high-quality rendering. Furthermore, we develop a differentiable depth peeling algorithm that utilizes the hardware rasterization pipeline to effectively optimize and efficiently render the proposed model. In our experiments, we show that 4K4D not only achieves state-of-the-art rendering quality but also exhibits a more than 30× increase in rendering speed (over 200FPS at 1080p on an RTX 3090).

However, our method still has some limitations. 4K4D cannot produce correspondences of points across frames, which are important for some downstream tasks. Moreover, the storage cost for 4K4D increases linearly with the number of video frames, so our method has difficulty in modeling long volumetric videos. How to model correspondences and reduce the storage cost for long videos could be two interesting problems for future works.

References

[1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*, pages 696–712. Springer, 2020.

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

[3] Benjamin Attal, Jia-Bin Huang, Michael Zollhöfer, Johannes Kopf, and Changil Kim. Learning neural light fields with ray-space embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19819–19829, 2022.

[4] Louis Bavoil and Kevin Myers. Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, 1:12, 2008.

[5] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 425–432, New York, NY, USA, 2001. Association for Computing Machinery.

[6] Dan Casas, Marco Volino, John Collomosse, and Adrian Hilton. 4d video textures for interactive character appearance. In *Computer Graphics Forum*, pages 371–380. Wiley Online Library, 2014.

[7] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM TOG*, 2013.

[8] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *arXiv*, 2022.

[9] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *ICCV*, 2021.

[10] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022.

[11] Wei Cheng, Ruixiang Chen, Wanqi Yin, Siming Fan, Keyu Chen, Honglin He, Huiwen Luo, Zhongang Cai, Jingbo Wang, Yang Gao, et al. Dna-rendering: A diverse neural actor repository for high-fidelity human-centric rendering. *arXiv preprint arXiv:2307.10173*, 2023.

[12] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. High-quality streamable free-viewpoint video. *ACM Transactions on Graphics (ToG)*, 34(4):1–13, 2015.

[13] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. In *Computer Graphics Forum*, pages 305–314. Wiley Online Library, 2012.

[14] Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, et al. Fusion4d: Real-time performance capture of challenging scenes. *ACM TOG*, 2016.

[15] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *ACM Siggraph Computer Graphics*, 22(4):65–74, 1988.

[16] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *CVPR*, June 2016.

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

[18] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021.

[19] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *SIGGRAPH*, 1996.

[20] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, light, and material decomposition from images using monte carlo rendering and denoising. *Advances in Neural Information Processing Systems*, 35:22856–22869, 2022.

[21] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM TOG*, 2018.

[22] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021.

[23] Anna Hilsmann, Philipp Fechteler, Wieland Morgenstern, Wolfgang Paier, Ingo Feldmann, Oliver Schreer, and Peter Eisert. Going beyond free viewpoint: creating animatable volumetric video of human performances. *IET Computer Vision*, pages 350–358, 2020.

[24] Mustafa Işık, Martin Rünz, Markos Georgopoulos, Taras Khakhulin, Jonathan Starck, Lourdes Agapito, and Matthias Nießner. Humanrf: High-fidelity neural radiance fields for humans in motion. *arXiv preprint arXiv:2305.06356*, 2023.

[25] Shubhendu Jena, Franck Multon, and Adnane Boukhayma. Neural mesh-based graphics. In *European Conference on Computer Vision*, pages 739–757. Springer, 2022.

[26] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *CVPR*, 2020.

[27] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM TOG*, 2016.

[28] Petr Kellnhofer, Lars C Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4287–4297, 2021.

[29] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4):1–14, 2023.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Computer Graphics Forum*, volume 40, pages 29–43. Wiley Online Library, 2021.

[32] Jonas Kulhanek and Torsten Sattler. Tetra-nerf: Representing neural radiance fields using tetrahedra. *arXiv preprint arXiv:2304.09987*, 2023.

[33] Kiriakos N Kutulakos and Steven M Seitz. A theory of shape by space carving. *International journal of computer vision*, 38:199–218, 2000.

[34] Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1440–1449, 2021.

[35] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH*, 1996.

[36] Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: Efficient sampling accelerates nerfs. *arXiv preprint arXiv:2305.04966*, 2023.

[37] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, and Zhaoyang Lv. Neural 3d video synthesis. *arXiv preprint arXiv:2103.02597*, 2021.

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

[39] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *CVPR*, 2021.

[40] Zhengqi Li, Qianqian Wang, Forrester Cole, Richard Tucker, and Noah Snavely. Dynibar: Neural dynamic image-based

rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4273–4284, 2023.

[41] Zhengqi Li, Wenqi Xian, Abe Davis, and Noah Snavely. Crowdsampling the plenoptic function. In *ECCV*, 2020.

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

[43] Haotong Lin, Sida Peng, Zhen Xu, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Efficient neural radiance fields for interactive free-viewpoint video. In *SIGGRAPH Asia Conference Proceedings*, 2022.

[44] Shanchuan Lin, Linjie Yang, Imran Saleemi, and Soumyadip Sengupta. Robust high-resolution video matting with temporal guidance. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 238–247, 2022.

[45] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3d supervision. *NeurIPS*, 2019.

[46] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. In *SIGGRAPH*, 2019.

[47] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021.

[48] Fan Lu, Yan Xu, Guang Chen, Hongsheng Li, Kwan-Yee Lin, and Changjun Jiang. Urban radiance field representation with deformable neural mesh primitives. *arXiv preprint arXiv:2307.10776*, 2023.

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

[50] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM TOG*, 2019.

[51] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[52] Claus Müller. *Spherical harmonics*, volume 17. Springer, 2006.

[53] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.

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

[55] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L Davidson, Sameh Khamis, Mingsong Dou, et al. Holoportation: Virtual 3d teleportation in real-time. In *UIST*, 2016.

[56] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *ICCV*, 2021.

[57] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021.

[58] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical report, Technical Report UUCS-98-019, Computer Science Department, University of Utah, 1998.

[59] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[60] Nikolay Patakin, Dmitry Senushkin, Anna Vorontsova, and Anton Konushin. Neural global illumination for inverse rendering. In *2023 IEEE International Conference on Image Processing (ICIP)*, pages 1580–1584. IEEE, 2023.

[61] Sida Peng, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Representing volumetric videos as dynamic mlp maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4252–4262, 2023.

[62] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9054–9063, 2021.

[63] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM TOG*, 2017.

[64] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *CVPR*, 2021.

[65] Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky, and Evgeny Burnaev. Npbg++: Accelerating neural point-based graphics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15969–15979, 2022.

[66] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[67] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, pages 14335–14345, 2021.

[68] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG)*, 41(4):1–14, 2022.

[69] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[70] Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. Tensor4d: Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering. *arXiv*, 2022.

[71] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *CVPR*, 2020.

[72] Dave Shreiner et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.

[73] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[74] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *Advances in Neural Information Processing Systems*, 34:19313–19325, 2021.

[75] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR*, 2019.

[76] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *NeurIPS*, 2019.

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

[78] Pratul P Srinivasan, Richard Tucker, Jonathan T Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 175–184, 2019.

[79] Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. Light field neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8269–8279, June 2022.

[80] Richard Szeliski and Polina Golland. Stereo matching with transparency and matting. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 517–524. IEEE, 1998.

[81] Suramya Tomar. Converting video formats with ffmpeg. *Linux journal*, 2006(146):10, 2006.

[82] Feng Wang, Sinan Tan, Xinghang Li, Zeyue Tian, and Huaping Liu. Mixed neural voxels for fast multi-view video synthesis. *arXiv preprint arXiv:2212.00190*, 2022.

[83] Liao Wang, Qiang Hu, Qihan He, Ziyu Wang, Jingyi Yu, Tinne Tuytelaars, Lan Xu, and Minye Wu. Neural residual radiance fields for streamably free-viewpoint videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 76–87, 2023.

[84] Liao Wang, Jiakai Zhang, Xinhang Liu, Fuqiang Zhao, Yanshun Zhang, Yingliang Zhang, Minye Wu, Jingyi Yu, and Lan Xu. Fourier plenoctrees for dynamic radiance field rendering in real-time. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13524–13534, 2022.

[85] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2021.

[86] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021.

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

[88] Minye Wu, Yuehao Wang, Qiang Hu, and Jingyi Yu. Multi-view neural human rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1682–1691, 2020.

[89] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023.

[90] Zeyu Yang, Hongye Yang, Zijie Pan, Xiatian Zhu, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. *arXiv preprint arXiv 2310.10642*, 2023.

[91] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *CVPR*, 2022.

[92] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021.

[93] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021.

[94] Tao Yu, Zerong Zheng, Kaiwen Guo, Jianhui Zhao, Qionghai Dai, Hao Li, Gerard Pons-Moll, and Yebin Liu. Doublefusion: Real-time capture of human performances with inner body shapes from a single depth sensor. In *CVPR*, 2018.

[95] Qiang Zhang, Seung-Hwan Baek, Szymon Rusinkiewicz, and Felix Heide. Differentiable point-based radiance fields for efficient view synthesis. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–12, 2022.

[96] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[97] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video

view interpolation using a layered representation. *ACM TOG*, 2004.

Figure A1. **Qualitative comparison on the NHR [88] dataset that contains 512×612 (and 384×512) images.**

Table A1. **Number of views and frames for each dataset's used sequences.** The *basketball* sequence of NHR [88] provides 72 views compared to the 56 views for the rest of the dataset.

Dataset	Sequence Count	Traininig View	Testing View	Frame Count
DNA-Rendering [11]	4	56	4	150
NHR (*sport*) [88]	3	52	4	100
NHR (*basketball*) [88]	1	68	4	100
ENeRF-Outdoor [43]	3	17	1	100
Neural3DV [38]	1	19	1	300

Table A2. **Quantitative comparison on the NHR [88] dataset.** This dataset includes 512×612 and 384×512 images. Metrics are averaged over all scenes.

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS
ENeRF [43]	30.765	0.954	0.052	10.432
IBRNet [85]	33.537	0.965	0.078	0.369
KPlanes [17]	32.933	0.958	0.101	1.979
MlpMaps [61]	32.203	0.953	0.080	30.303
DyNeRF [38]	30.872	0.943	0.117	0.192
Ours	33.743	0.973	0.045	237.919

Appendix

A. Dataset Settings

In Tab. A1, we list the detailed sequence count, view count, and frame count for each of the datasets used in our experiments. For image-based baselines [43, 85] and our method, the training view is also the source view for the IBR process. No test views are provided as source views.

B. Additional Experiments

B.1. Additional Comparisons on NHR [88]

In Tab. A2 and Fig. A1, we provide quantitative and qualitative comparisons on the NHR dataset [88] with multiple baselines. KPlanes [17] and DyNeRF [38] can only produce blurry images on this dataset that include fast motions, and their rendering speed is also limited (2.0 and 0.2 FPS). ENeRF [43] and MlpMaps [61] can be rendered at interactive frame rates with moderate resolution, but our method can produce higher quality results at much higher frame rates (30FPS *vs.* 238FPS).

B.2. Visualization of SH color

In Fig. A2, we visualize the SH color $\mathbf{c}_{sh}(\mathbf{s}, \mathbf{d})$ on 5 rotating views of the *0013_01* sequence of DNA-Rendering [11]. The SH model provides the fine-level appearance that enables the continuous view-dependent effects.

B.3. Comparisons with 3DGS [29]

We perform additional comparisons with 3DGS [29] on the first frame of the *actor1_4* sequence of the ENeRF-Outdoor [43] dataset in Tab. A3 and Fig. A3. The storage cost of our method contains both the file sizes (in MB) of the trained model and source images. After the precomputation in Sec. 3.4, the main memory usage of our method is 100 MB per frame, which is still smaller than the disk file size

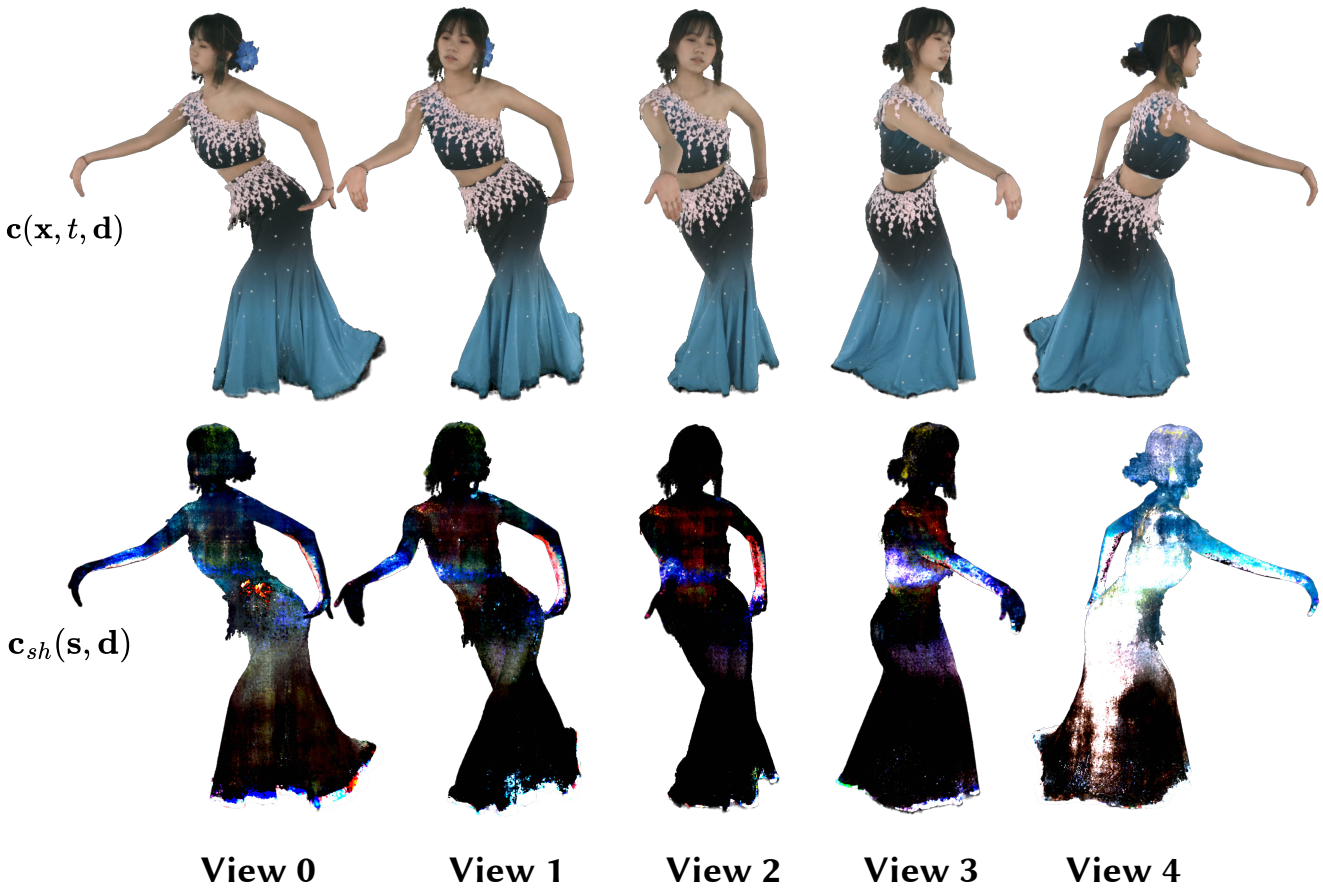

Figure A2. **Visualization of $\mathbf{c}(\mathbf{x}, t, \mathbf{d})$ and $\mathbf{c}_{sh}(\mathbf{s}, \mathbf{d})$ on 5 rotating views of the *0013_01* sequence of DNA-Rendering [11].** The view-dependent SH color $\mathbf{c}_{sh}$ compensates the high-quality but discrete IBR color $\mathbf{c}_{ibr}$. We increase the brightness of $\mathbf{c}_{sh}$ for a clearer visualization. Details of the implementation can be found in Sec. 3.1.

Table A3. **Quantitative comparison on the first frame of the all three sequences of the ENeRF-Outdoor [43] dataset.** The dataset contains 18 images of 960 × 540 resolution. "Storage" indicates the disk file size of the trained models (and source images for our method).

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS	Storage	Training
Gaussian [29]	21.633	0.608	0.349	88.355	715 MB	0.5 hour
Ours	26.544	0.907	0.145	148.581	16.0 MB	1.5 hour

of 3DGS. Our method is trained for 5,000 iterations, and 3DGS is trained for 30,000 iterations (their default settings) using their official source code. 3DGS renders slower than our method because too many points were generated during their training process. Qualitatively, 3DGS overfits the first frame and fails to generalize to novel views, as indicated by the last two rows of Fig. A3.

Compared to training a 3DGS for every frame, our method is superior in the following ways. First, the storage cost for 3DGS is too large for even a 100-frame video (715 MB per frame), while our method maintains a reasonable 2 MB per frame storage overhead (Sec. 5.2). Thanks to the implicit compression of our 4D feature grid and IBR model Sec. 5.2, our method better utilizes the temporal redundancy of the dynamic 3D scene. The optimization and precomputation of our method can be viewed as a form of compression and decompression, where we encode and decode the dynamic 3D scene using 4D feature grids, network weights, and source videos. Second, the hybrid image-based appearance model (Sec. 3.1) of our method is more expressive than the spherical harmonics utilized by 3DGS, thus achieving higher rendering quality as shown by Fig. A3 and Tab. A3.

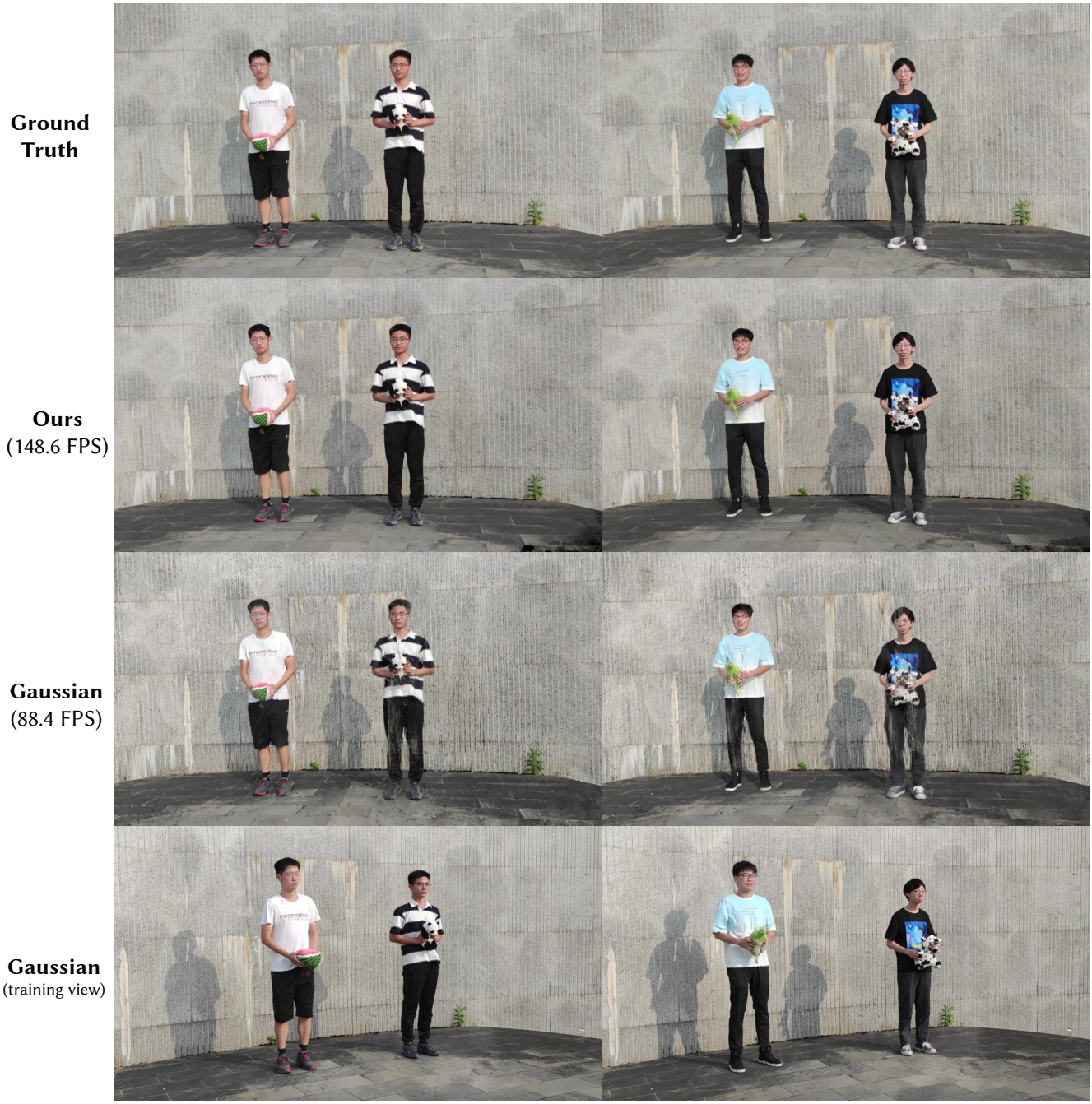

Figure A3. **Qualitative comparison on the first frame of the *actor1_4* sequence of ENeRF-Outdoor [43] dataset.** The first frame contains 18 images of 960 × 540 resolution. 3D Gaussian Splatting [29] overfits the training view as indicated by the last two rows.