HexPlane: A Fast Representation for Dynamic Scenes

Ang Cao Justin Johnson

University of Michigan, Ann Arbor

Abstract

Modeling and re-rendering dynamic 3D scenes is a challenging task in 3D vision. Prior approaches build on NeRF and rely on implicit representations. This is slow since it requires many MLP evaluations, constraining real-world applications. We show that dynamic 3D scenes can be explicitly represented by six planes of learned features, leading to an elegant solution we call HexPlane. A HexPlane computes features for points in spacetime by fusing vectors extracted from each plane, which is highly efficient. Pairing a HexPlane with a tiny MLP to regress output colors and training via volume rendering gives impressive results for novel view synthesis on dynamic scenes, matching the image quality of prior work but reducing training time by more than $100\times$*. Extensive ablations confirm our HexPlane design and show that it is robust to different feature fusion mechanisms, coordinate systems, and decoding mechanisms. HexPlanes are a simple and effective solution for representing 4D volumes, and we hope they can broadly contribute to modeling spacetime for dynamic 3D scenes.* [1]

1. Introduction

Reconstructing and re-rendering 3D scenes from a set of 2D images is a core vision problem which can enable many AR/VR applications. The last few years have seen tremendous progress on reconstructing *static* scenes, but this assumption is restrictive: the real world is *dynamic*, and in complex scenes motion is the norm, not the exception.

Many current approaches for representing dynamic 3D scenes rely on *implicit* representations, building on NeRF [32]. They train a large multi-layer perceptron (MLP) that inputs the position of a point in space and time, and outputs either the color of the point [23] or a *deformation* to a canonical static scene [13, 22, 38, 39, 43]. In either case, rendering images from novel views is expensive since each generated pixel requires many MLP evaluations. Training is similarly slow, requiring up to days of GPU-time to model a single dynamic scene; this computational bottleneck prevents these methods from being widely applied.

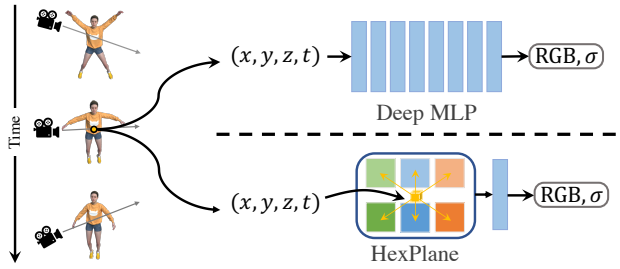


Figure 1. **HexPlane for Dynamic 3D Scenes.** Instead of regressing colors and opacities from a deep MLP, we explicitly compute features for points in spacetime via HexPlane. Pairing with a tiny MLP, it allows above $100\times$ speedups with matching quality.

Several recent methods for modeling *static* scenes have demonstrated tremendous speedups over NeRF through the use of *explicit* and *hybrid* methods [7, 33, 51, 62]. These methods use an explicit spatial data structure that stores explicit scene data [12, 62] or features that are decoded by a tiny MLP [7, 33, 51]. This decouples a model's *capacity* from its *speed*, and allows high-quality images to be rendered realtime [33]. While effective, these methods have thus far been applied only to static scenes.

In this paper we aim to design an explicit representation for dynamic 3D scenes, building on similar advances for static scenes. To this end we design a *spatial-temporal* data structure that stores scene data. It must overcome two key technical challenges. First is *memory usage*. We must model all points in both space and time; naïvely storing data in a dense 4D grid would scale with the fourth power of grid resolution which is infeasible for large scenes or long durations. Second is *sparse observations*. Moving a single camera through a static scene can give views that densely cover the scene; in contrast, moving a camera through a dynamic scene gives just one view per timestep. Treating timesteps independently may give insufficient scene coverage for high-quality reconstruction, so we must instead share information across timesteps.

We overcome these challenges with our novel HexPlane architecture. Inspired by factored representations for static

[1] Project page: https://caoang327.github.io/HexPlane.

scenes [5, 7, 40], a HexPlane decomposes a 4D spacetime grid into six *feature planes* spanning each pair of coordinate axes (*e.g.* XY, ZT). A HexPlane computes a feature vector for a 4D point in spacetime by projecting the point onto each feature plane, then aggregating the six resulting feature vectors. The fused feature vector is then passed to a tiny MLP which predicts the color of the point; novel views can then be rendered via volume rendering [32].

Despite its simplicity, a HexPlane provides an elegant solution to the challenges identified above. Due to its factored representation, a HexPlane's memory footprint only scales quadratically with scene resolution. Furthermore, each plane's resolution can be tuned independently to account for scenes requiring variable capacity in space and time. Since some planes rely only on *spatial* coordinates (*e.g.* XY), by construction a HexPlane encourages sharing information across disjoint timesteps.

Our experiments demonstrate that HexPlane is an effective and highly efficient method for novel view synthesis in dynamic scenes. On the challenging Plenoptic Video Dataset [23] we match the image quality of prior work but improve training time by $>100\times$; we also outperform prior approaches on a monocular video dataset [43]. Extensive ablations validate our HexPlane design and demonstrate that it is robust to different feature fusion mechanisms, coordinate systems (rectangular vs spherical), and decoding mechanisms (spherical harmonics vs MLP).

HexPlane is a simple, explicit, and general representation for dynamic 3D scenes. It makes minimal assumptions about the underlying scene, and does not rely on deformation fields or category-specific priors. Besides improving and accelerating view synthesis, we hope that HexPlane will be useful for a broad range of research in dynamic scenes.

2. Related Work

Neural Scene Representations. Using neural networks to implicitly represent 3D scenes [29, 36, 48, 49, 52, 57] has drawn much recent attention. NeRF [32] and its variants [2, 3, 30, 34, 53, 54, 61, 66] have achieved impressive results on novel view synthesis [8, 57, 63] and have many applications ncluding 3D reconstruction [28, 52, 64, 69, 72], semantic segmentation [19, 44, 71], generative model [5, 6, 9, 35, 46], 3D content creation [1, 17, 37, 42, 55, 65].

Implicit neural representations show impressive rendering quality but suffer from slow rendering speeds since many expensive MLP evaluations are needed to render each pixel. To overcome this issue, many recent papers propose *hybrid* representations that combine a fast explicit scene representation with learnable neural network components [4, 5, 7, 12, 40, 51, 60], providing significant spedups over purely implicit methods. Many types of explicit representations have been explored including sparse voxels [12, 51], triplanes [5, 40], point clouds [60], and tensor factorizations [7]. However, these approaches assume static 3D scenes; hybrid representations for dynamic scenes are under-explored. This paper proposes an explicit model for radiance fields in dynamic scenes, hugely accelerating prior methods that rely on fully implicit methods.

Neural Rendering for Dynamic Scenes. Representing dynamic scenes by neural radiance fields is an essential extension of NeRF and empowers many real-world applications [21, 41, 50, 59, 70]. One line of research represents dynamic scenes by extending NeRF with an additional time dimension (T-NeRF) or additional latent code [13, 23, 58]. With the capability to represent any deformation or typology changes, these methods result in a severely underconstrained problem, allowing for trivial, non-plausible solutions. Additional supervisions like depths, optical flows, or dense multi-view observations are generally needed to get decent results. Another line of research employs individual MLPs to represent a deformation field and a canonical field [38, 39, 43], where the canonical field represents a static scene, and the deformation field learns the coordinate maps to the canonical space across times. Instead of improving performance, the main target of our paper is to propose an explicit and compact representation that could replace T-NeRF or deformation field MLP.

Accelerating NeRFs Our work is also related to NeRF accelerations. Considerable works have been proposed to accelerate NeRF at diverse stages. Some methods improve inference speeds of trained NeRFs by by reformulating the computation [15, 45], using precomputation [16, 62]. Other methods reduce the training times by learning a generalizable model [8, 56]. Recently, some methods significantly improve rendering speeds during both training and inference by reducing computations with hybrid explicit-implicit representations [5, 7, 12, 24, 33, 51]. With a similar idea, our method achieves tremendous acceleration during training and inference by explicitly representing dynamic fields.

Very recently, Tineuvox [11] has been proposed to accelerate dynamic NeRFs [39, 43], while it is distinct from ours in several aspects. Tineuvox [11] follows the idea of D-NeRF [43] and represents dynamic 3D scenes by a deformation field with a canonical static 3D space, while our method is designed for general spatial-temporary fields without such deformation assumption. Technically, it replaces D-NeRF's [43] MLP with a sparse voxel grid [51] to represent static canonical space for accelerations, which grid is a fast explicit representation for static 3D scenes with highly optimized cuda kernels. However, it still uses MLPs to represent deformation fields. On the contrary, our paper proposes an explicit representation directly for dynamic 3D fields, which is not explored by previous work.

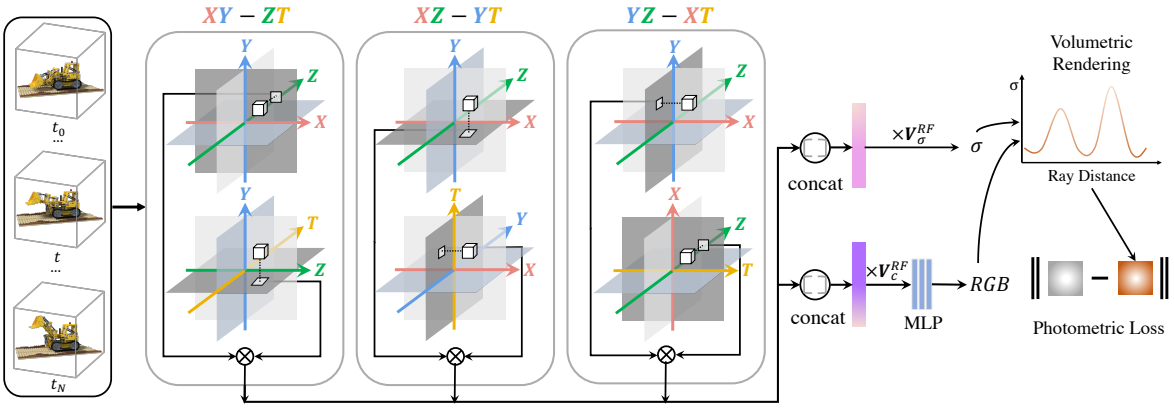


Figure 2. **Method Overview.** HexPlane contains six feature planes spanning each pair of coordinate axes (e.g. XY , ZT). To compute features of points in spacetime, it multiplies feature vectors extracted from paired planes and concatenated multiplied results into a single vector, which are then multiplied by $\mathbf{V}^{RF}$ for final results. RGB colors are regressed from point features using a tiny MLP and images are synthesized via volumetric rendering. HexPlane and the MLP are trained by photometric loss between rendered and target images.

3. Method

Given a set of posed and timestamped images of a dynamic scene, we aim to fit a model to the scene that allows rendering new images at novel poses and times. Like NeRF [32], a model gives color and opacity for points in spacetime; images are rendered via differentiable volumetric rendering along rays. The model is trained using photometric loss between rendered and ground-truth images.

Our main contribution is a new *explicit* representation for dynamic 3D scenes, which we combine with a small *implicit* MLP to achieve novel view synthesis in dynamic scenes. An input spacetime point is used to efficiently query the explicit representation for a feature vector. A tiny MLP receives the feature along with the point coordinates and view direction and regresses an output RGB color for the point. Figure 2 shows an overview of the model.

Designing an explicit representation for dynamic 3D scenes is challenging. Unlike static 3D scenes which are often modeled by point clouds, voxels, or meshes, explicit representations for dynamic scenes have been underexplored. We show how the key technical challenges of *memory usage* and *sparse observations* can be overcome by our simple HexPlane representation.

3.1. 4D Volumes for Dynamic 3D Scenes

A dynamic 3D scene could be naïvely represented as a 4D volume $\mathbf{D}$ comprising independent static 3D volumes per time step $\{\mathbf{V_1}, \mathbf{V_2}, \cdots, \mathbf{V_T}\}$. However this design suffers from two key problems. First is *memory consumption*: a naïve 4D volume is very memory intensive, requiring $O(N^3TF)$ space where N, T, and F are the spatial resolution, temporal resolution, and feature size. Storing a volume of RGB colors (F=3) with N=512, T=32 in `float32` format takes 48GB of memory.

The second problem is *sparse observations*. A single camera moving through a static scene can capture dozens or hundreds of images. In dynamic scenes capturing multiple images per timestep requires multiple cameras, so we typically have only a few views per timestep; these sparse views are insufficient for independently modeling each timestep, so we must share information between timesteps.

We reduce *memory consumption* using *factorization* [5, 7] which has been previously applied to 3D volumes. We build on TensoRF [7] which decomposes a 3D volume $\mathbf{V} \in \mathbb{R}^{XYZF}$[1] as a sum of vector-matrix outer products:

$$\mathbf{V} = \sum_{r=1}^{R_1} \mathbf{M}_r^{XY} \circ \mathbf{v}_r^{Z} \circ \mathbf{v}_r^{1} + \sum_{r=1}^{R_2} \mathbf{M}_r^{XZ} \circ \mathbf{v}_r^{Y} \circ \mathbf{v}_r^{2} + \sum_{r=1}^{R_3} \mathbf{M}_r^{ZY} \circ \mathbf{v}_r^{X} \circ \mathbf{v}_r^{3} \tag{1}$$

where $\circ$ is outer product; $\mathbf{M}_r^{XY} \circ \mathbf{v}_r^{Z} \circ \mathbf{v}_r^{1}$ is a low-rank component of $\mathbf{V}$; $\mathbf{M}_r^{XY} \in \mathbb{R}^{XY}$ is a matrix spanning the X and Y axes, and $\mathbf{v}^{Z} \in \mathbb{R}^{Z}, \mathbf{v}_r^{i} \in \mathbb{R}^{F}$ are vectors along the Z and F axes. R_1, R_2, R_3 are the number of low-rank components. When $R = R_1 + R_2 + R_3 \ll N$, this design reduces memory usage from $O(N^3TF)$ to $O(RN^2T)$.

3.2. Linear Basis for 4D Volume

Factorization helps reduce memory usage, but factoring an independent 3D volume per timestep still suffers from sparse observations and does not share information across time. To solve this problem, we can represent the 3D volume $\mathbf{V_t}$ at time t as the weighted sum of a set of shared 3D basis volumes $\{\hat{\mathbf{V}}_1, \ldots, \hat{\mathbf{V}}_{R_t}\}$; then

$$\mathbf{V_t} = \sum_{i=1}^{R_t} \mathbf{f}(t)_i \cdot \hat{\mathbf{V}}_i \tag{2}$$

[1] To simplify notation, we write $\mathbb{R}^{X \times Y}$ as $\mathbb{R}^{XY}$ in this paper.

where $\cdot$ is a scalar-volume product, R_t is the number of shared volumes, and $\mathbf{f}(t) \in \mathbb{R}^{R_t}$ gives weights for the shared volumes as a function of t. Shared volumes allow information to be shared across time. In practice each $\hat{\mathbf{V}}_i$ is represented as a TensoRF as in Equation 1 to save memory.

Unfortunately, we found in practice (and will show with experiments) that shared volumes are still too costly; we can only use small values for R_t without exhausting GPU memory. Since each shared volume is a TensoRF, it has its own independent $\mathbf{M}_r^{XY}, \mathbf{v}_r^Z$, *etc*.; we can further improve efficiency by sharing these low-rank components across all shared volumes. The 3D volume $\mathbf{V}_t$ at time t is then

$$\begin{aligned}\mathbf{V}_t = &\sum_{r=1}^{R_1} \mathbf{M}_r^{XY} \circ \mathbf{v}_r^Z \circ \mathbf{v}_r^1 \cdot \mathbf{f}^1(t)_r + \sum_{r=1}^{R_2} \mathbf{M}_r^{XZ} \circ \mathbf{v}_r^Y \circ \mathbf{v}_r^2 \cdot \mathbf{f}^2(t)_r \\ &+ \sum_{r=1}^{R_3} \mathbf{M}_r^{ZY} \circ \mathbf{v}_r^X \circ \mathbf{v}_r^3 \cdot \mathbf{f}^3(t)_r\end{aligned} \tag{3}$$

where each $\mathbf{f}^i(t) \in \mathbb{R}^{R_i}$ gives a vector of weights for the low-rank components at each time t.

In this formulation, $\mathbf{f}^i(t)$ captures the model's depdendence on time. The correct mathematical form for $\mathbf{f}^i(t)$ is not obvious. We initially framed $\mathbf{f}^i(t)$ as a learned combination of sinusoidal or other fixed basis functions, with the hope that this could make periodic motion easier to learn; however we found this inflexible and hard to optimize. $\mathbf{f}^i(t)$ could be an arbitrary nonlinear mapping, represented as an MLP; however this would be slow. As a pragmatic tradeoff between flexibility and speed, we represent $\mathbf{f}^i(t)$ as a learned piecewise linear function, implemented by linearly interpolating along the first axis of a learned $T \times R_i$ matrix.

3.3. HexPlane Representation

Equation 3 fully decouples the spatial and temporal modeling of the scene: $\mathbf{f}^i(t)$ models time and other terms model space. However in real scenes space and time are entangled; for example a particle moving in a circle is difficult to model under Equation 3 since its x and y positions are best modeled *separately* as functions of t. This motivates us to replace $\mathbf{v}_r^Z \cdot \mathbf{f}^1(t)_r$ in Equation 3 with a joint function of t and z, similarly represented as a piecewise linear function; this can be implemented by bilinear interpolation into a learned tensor of shape $Z \times T \times R_1$. Applying the same transform to all similar terms then gives our HexPlane representation, which represents a 4D feature volume $V \in \mathbb{R}^{XYZTF}$ as:

$$\begin{aligned}\mathbf{D} = &\sum_{r=1}^{R_1} \mathbf{M}_r^{XY} \circ \mathbf{M}_r^{ZT} \circ \mathbf{v}_r^1 + \sum_{r=1}^{R_2} \mathbf{M}_r^{XZ} \circ \mathbf{M}_r^{YT} \circ \mathbf{v}_r^2 \\ &+ \sum_{r=1}^{R_3} \mathbf{M}_r^{YZ} \circ \mathbf{M}_r^{XT} \circ \mathbf{v}_r^3\end{aligned} \tag{4}$$

where each $\mathbf{M}_r^{AB} \in \mathbb{R}^{AB}$ is a learned plane of features. This formulation displays a beautiful symmetry, and strikes a balance between representational power and speed.

We can alternatively express a HexPlane as a function $\mathbf{D}$ which maps a point (x, y, z, t) to an F-dimensional feature:

$$\begin{aligned}D(x,y,z,t) = &(\mathbf{P}_{xy\bullet}^{XYR_1} \odot \mathbf{P}_{zt\bullet}^{ZTR_1})\mathbf{V}^{R_1F} \\ &+(\mathbf{P}_{xz\bullet}^{XZR_2} \odot \mathbf{P}_{yt\bullet}^{YTR_2})\mathbf{V}^{R_2F} \\ &+(\mathbf{P}_{yz\bullet}^{YZR_3} \odot \mathbf{P}_{xt\bullet}^{XTR_3})\mathbf{V}^{R_3F}\end{aligned} \tag{5}$$

where $\odot$ is an elementwise product; the superscript of each bold tensor represents its shape, and $\bullet$ in a subscript represents a slice so each term is a vector-matrix product. $\mathbf{P}^{XYR_1}$ stacks all $\mathbf{M}_r^{XY}$ to a 3D tensor, and $\mathbf{V}^{R_1F}$ stacks all $\mathbf{v}_r^1$ to a 2D tensor; other terms are defined similarly. Coordinates x, y, z, t are real-valued, so subscripts denote bilinear interpolation. This design reduces memory usage to $O(N^2R + NTR + RF)$.

We can stack all $\mathbf{V}^{R_iF}$ into $\mathbf{V}^{RF}$ and rewrite Eq 5 as

$$[\mathbf{P}_{xy\bullet}^{XYR_1} \odot \mathbf{P}_{zt\bullet}^{ZTR_1}; \mathbf{P}_{xz\bullet}^{XZR_2} \odot \mathbf{P}_{yt\bullet}^{YTR_2}; \mathbf{P}_{yz\bullet}^{YZR_3} \odot \mathbf{P}_{xt\bullet}^{XTR_3}]\mathbf{V}^{RF} \tag{6}$$

where ; concatenates vectors. As shown in Figure 2, a HexPlane comprises three pairs of feature planes; each pair has a spatial and a spatio-temporal plane with orthogonal axes (*e.g.* XY/ZT). Querying a HexPlane is fast, requiring just six bilinear interpolations and a vector-matrix product.

3.4. Optimization

We represent dynamic 3D scenes using the proposed HexPlane, which is optimized by photometric loss between rendered and target images. For point (x, y, z, t), its opacity and appearance feature are quired from HexPlane, and the final RGB color is regressed from a tiny MLP with appearance feature and view direction as inputs. With points' opacities and colors, images are rendered via volumetric rendering. The optimization objective is:

$$\mathcal{L} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} \|C(\mathbf{r}) - \hat{C}(\mathbf{r})\|_2^2 + \lambda_{\text{reg}} \mathcal{L}_{\text{reg}} \tag{7}$$

where $\mathcal{L}_{\text{reg}}$ is regularization and λ_{reg} is associated weight; $\mathcal{R}$ is the set of image rays and $C(\mathbf{r}), \hat{C}(\mathbf{r})$ are rendered and GT colors of ray $\mathbf{r}$.

Color Regression. To save computations, we query points' opacities directly from one HexPlane, and query appearance features of points with high opacities from another separate HexPlane. Queried features and view directions are fed into a tiny MLP for RGB colors. An MLP-free design is also feasible with spherical harmonics coefficients as features.

Regularizer. Dynamic 3D reconstruction is a severely ill-posed problem, needing strong regularizers. We apply Total Variational (TV) loss on planes to force the spatial-temporal continuity, and depth smooth loss in [34] to reduce artifacts. We apply different weights of TV loss along spatial and temporal axes and found a larger weight along the temporal axis is helpful for very sparse observations. We also utilize dist loss from [3] for real-world monocular videos.

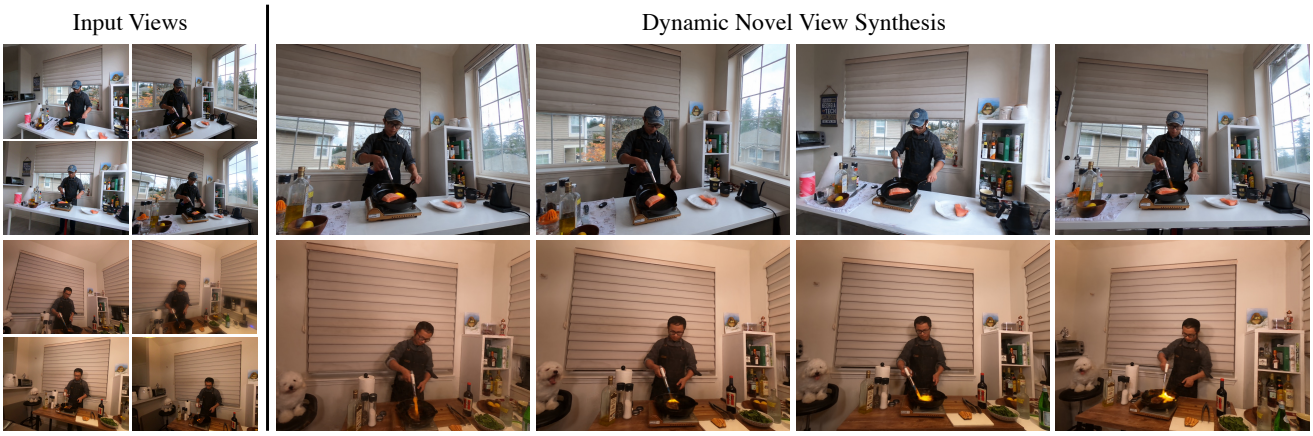


Figure 3. **High-Quality Dynamic Novel View Synthesis on Plenoptic Video dataset [22].** The proposed HexPlane could effectively represent dynamic 3D scenes with complicated motions and render high-quality results with faithful details at various timesteps and unseen viewpoints. We show several samples of input video sequences and synthesis results using a cyclic camera trajectory.

Coarse to Fine Training. A coarse-to-fine scheme is also employed like [7, 62], where the resolution of grids gradually grows during training. This design accelerates the training and provides an implicit regularization on nearby grids.

Emptiness Voxel. We keep a tiny 3D voxel indicating the emptiness of scene regions and skip points in empty regions. Since many regions are empty, it is helpful for acceleration. To get this voxel, we evaluate points' opacities across time steps and reduce them to a single voxel with maximum opacities. Although keeping several voxels for various time intervals improves speeds, we only keep one for simplicity.

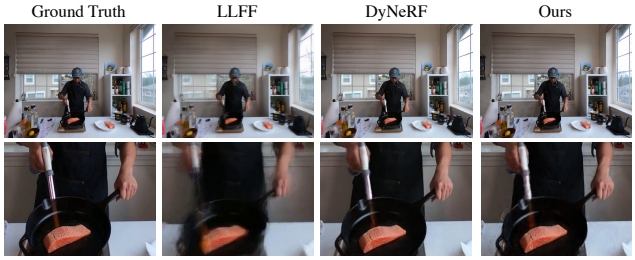


Figure 4. **Visual Comparison of Synthesis Results.** Since DyNeRF [22] model is not publicly available, we compare our results to images provided the paper. With visually similar results, our proposed HexPlane is over 100 $\times$ faster than DyNeRF.

4. Experiments

We propose HexPlane, an explicit representation of dynamic 3D scenes. To test its capacity, we use it for dynamic novel view synthesis on challenging datasets, comparing its performance and speed with the SOTA methods. Extensive ablations are studied to introspect its properties. Moreover, we show an extension of HexPlane to model unbound scenes and test its ability on iPhone videos.

4.1. Dynamic Novel View Synthesis Results

We use two datasets with different settings for a comprehensive evaluation: *Plenoptic Video dataset* [22], a high-resolution multi-camera dataset with highly challenging dynamic contents and visual details; *D-NeRF dataset* [43], a monocular video dataset with synthetic objects. *Plenoptic Video dataset* evaluates the representation ability of HexPlane for long videos with complicated motions and fine visual details. *D-NeRF dataset* challenges its ability to cope with monocular videos and very sparse observations.

Plenoptic Video dataset [22] is a real-world dataset captured by a multi-view camera system using 21 GoPro at 2028 $\times$ 2704 (2.7K) resolution and 30 FPS. Each scene contains 19 time-synchronized videos of 10 seconds in length, where 18 videos are used for training and 1 for evaluation. This dataset is suitable to test the representation ability of HexPlane since it contains various complicated and challenging dynamic contents, including objects of high specularity, translucency, and transparency; topology changes; self-casting moving shadows; fire flame and strong view-dependent effects for moving objects; and so on.

For a fair comparison, we follow the same training and evaluation pipelines as DyNeRF [22] with slight changes due to GPU resources. [22] trains its model on 8 V100 GPUs for a week, with 24576 batch size for 650K iterations. We train our model on a single 16GB V100 GPU, with 4096 batch size and the same iteration numbers, which is 6$\times$ fewer sampling. We follow the same importance sampling design and hierarch training as [22], with 512 spatial grid sizes and 300 time grid size. The scene is in NDC [32].

As shown in Figure 3, HexPlane could give high-quality dynamic novel view synthesis results across times and viewpoints. It could faithfully model real-world scenes with complicated motions and challenging appearances like flames, demonstrating its representation capacity.

Table 1. **Quantitative Comparisons on Plenoptic Video dataset [22].** We report synthesis quality, training times (measured in GPU hours) with speedups relative to DyNeRF [22]. With 224× speedups, HexPlane†with fewer training iterations has comparable quantitative results to DyNeRF. And HexPlane trained with the same iterations noticeably outperforms DyNeRF. Baseline methods are evaluated on a particular scene, and we also report average results on all public scenes (*-all*). Best and Second results are in highlight.

Model	Steps	PSNR↑	D-SSIM↓	LPIPS↓	JOD ↑	Training Time↓	Speeds-up ↑
Neural Volumes [25]	-	22.800	0.062	0.295	6.50	-	-
LLFF [31]	-	23.239	0.076	0.235	6.48	-	-
NeRF-T [22]	-	28.449	0.023	0.100	7.73	-	-
DyNeRF [22]	650k	29.581	0.020	0.099	8.07	1344h	1×
HexPlane	650k	29.470	0.018	0.078	8.16	12h	112×
HexPlane†	350k	29.182	0.021	0.089	8.08	6h	224 ×
HexPlane-all	650k	31.705	0.014	0.075	8.47	12h	112 ×
HexPlane†-all	350k	31.615	0.015	0.079	8.41	6h	224×

Table 2. **Quantitative Results on D-NeRF dataset [43].** Without deformation, HexPlane has comparable or better results compared to other deformation-based methods, and is noticeably faster.

Model	Deform.	PSNR↑	SSIM ↑	LPIPS ↓	Training Time↓
T-NeRF [43]		29.51	0.95	0.08	-
D-NeRF [43]	✓	30.50	0.95	0.07	20 hours
TiNeuVox-S [11]	✓	30.75	0.96	0.07	12m 10s
TiNeuVox-B [11]	✓	**32.67**	**0.97**	**0.04**	49m 46s
HexPlane (ours)		31.04	**0.97**	**0.04**	**11m 30s**

Quantitative comparisons with SOTA methods are in Table 1, with baseline numbers quoted from [22] paper. PSNR, Structural dissimilarity index measure (DSSIM), Perceptual quality measure LPIPS [67] and video visual difference measure Just-Objectionable-Difference (JOD) [27] are evaluated to measure synthesis quality comprehensively. Besides reporting results on the "flame salmon" scene like [22], we also report average results on all public scenes except unsynchronized one, referred to *HexPlane-all*. We also train a model with less training iterations as *HexPlane†*

In Table 1, HexPlane outperforms DyNeRF in all metrics except PSNR, indicating a comparable or even better synthesis quality. DyNeRF uses a giant MLP and per-frame latent codes to represent dynamic scenes, which is slow because of tremendous MLP evaluations. With the same training iterations, HexPlane is above 100 × faster than DyNeRF, illustrating the advantages of using explicit representations. Trained with fewer iterations, HexPlane†has comparable results with DyNeRF while even much faster.

Explicit representations generally require huge memories at the price of their fast speeds since they store features explicitly. HexPlane takes 200MB for the entire model, which is relatively compact for most GPUs. Considering its fast speed, we believe it is an appealing tradeoff.

Since the model of DyNeRF is not publicly available, it is hard to compare the visual results directly. We download images from the original paper and find the most matching images in our results, which are compared in Figure 4.

D-NeRF dataset [43] is a monocular video dataset with 360° observations for synthetic objects. Dynamic 3D reconstruction for monocular video is challenging since only one observation is available each time. Current SOTA methods for monocular video usually have a deformation field and a static canonical field, where points in dynamic scenes are mapped to positions in the canonical field. The mapping (deformation) is represented by another MLP.

The underlying assumption of deformation field design is that there are no topology changes, which does not always hold in the real world while holding in this dataset. Again, to keep HexPlane general enough, we do not assume deformation, which is the same as T-NeRF. We use this dataset to validate the ability to work with monocular observations.

We show quantitative results in Table 2. For fairness, all training times are re-measured on the same 2080TI GPU. Our HexPlane distinctly outperforms other methods even without introducing the deformation field, demonstrating the inherent ability to deal with sparse observations due to the shared basis. Again, our method is hundreds of times faster than MLP-based designs like D-NeRF and T-NeRF.

Tineuvox [11] is a recent work for accelerating D-NeRF, replacing canonical space MLP with a highly-optimized sparse voxel Cuda kernel and keeping an MLP to represent deformation. Therefore, it still uses explicit representation for static scenes while our target is dynamic scenes. Without any custom Cuda kernels, our method is faster and better than its light version and achieves the same LPIPS and SSIM as its bigger version, which takes longer time to train.

4.2. Ablations and Analysis

We run deep introspections to HexPlane by answering questions with extensive ablations. Ablations are conducted mainly on D-NeRF dataset because of efficiency.

How does HexPlane compare to others? We compare HexPlane with other designs mentioned in the Method Section in Table 3, where each method has various basis numbers # Basis: (1). *Volume Basis* represents 4D volumes as

Table 3. **Quantitative Results for Different Factorizations.** Various designs are evaluated on D-NeRF dataset with different R (basis number). HexPlane achieves the best quality and speed among all methods. Each method's best results are in highlight.

Model	R	PSNR↑	SSIM↑	LPIPS↓	Training Time↓
Volume Basis	8	30.460	0.965	0.045	18m 04s
	12	30.587	0.966	0.043	24m 06s
	16	30.631	0.967	0.042	29m 20s
VM-T	24	30.329	0.962	0.051	14m 36s
	48	30.657	0.965	0.048	15m 58s
	96	30.744	0.966	0.045	17m 03s
CP Decom.	48	28.370	0.942	0.083	10m 31s
	96	29.371	0.951	0.070	11m 03s
	192	30.086	0.957	0.063	11m 33s
	384	30.302	0.959	0.059	13m 06s
HexPlane	24	30.886	0.966	0.042	10m 27s
	48	31.042	0.968	0.039	11m 30s

Table 4. **Ablations on Feature Planes Designs.** We remove and swap HexPlane's planes and show results on D-NeRF dataset.

Model	PSNR↑	SSIM↑	LPIPS↓	Training Time↓
Spatial Planes	20.369	0.879	0.148	9m 02s
Spatial-Temporal Planes	21.112	0.879	0.148	9m 29s
DoublePlane (XY-ZT)	30.370	0.961	0.054	8m 04s
HexPlane-Swap	28.562	0.954	0.056	11m 44s
HexPlane	31.042	0.968	0.039	11m 30s

weighted summation of a set of shared 3D volumes as Eq 2, whcih 3D volume is represented as Eq 1; (2). *VMT* (vector, matrix and time) uses Eq 3 representing 4D volumes; (3). *CP Decom.* (CANDECOMP Decomposition) following [7], which represents 4D volumes using a set of vectors for each axis. Implementation details are in Supp.

HexPlane gives optimal performance among all methods, illustrating the advantages of spatial-temporal planes. Compared to other methods, spatial-temporal planes allow HexPlane to model motions effectively with small basis number R, leading to higher efficiency as well. Increasing R used for representation leads to better results while also resulting in more computations. We also notice that an unsuitable large R will lead to overfitting problem, which instead harms synthesis quality on novel views.

Could variants of HexPlane work? HexPlane has excellent symmetry as it contains all pairs of coordinate axes. By breaking this symmetry, we evaluate other variants in Table 4. *Spatial Planes* only have three spatial planes: $\mathbf{P}^{XY}, \mathbf{P}^{XZ}, \mathbf{P}^{YZ}$, and *Spatial-Temporal Planes* contain the left three spatial-temporal planes; *DoublePlane* contains only one group of paired planes, i.e. $\mathbf{P}^{XY}, \mathbf{P}^{ZT}$; *HexPlane-Swap* groups planes with repeated axes like $\mathbf{P}^{XY}, \mathbf{P}^{XT}$. We report their performance and speeds.

As shown in the table, neither *Spatial Planes* nor *Spatial-Temporal Planes* could represent dynamic scenes alone, indicating both are essential for representations. *HexPlane-Swap* achieves inferior results since its axes are not complementary, losing features from the particular axis. *DoublePlane* is less effective than HexPlane since HexPlane contains more comprehensive spatial-temporal modes.

Table 5. **Ablations on Feature Fusions Designs.** We show results with various fusion designs on D-NeRF dataset. HexPlane could work with other fusion mechanisms, showing its robustness.

Fusion-One	Fusion-Two	PSNR↑	SSIM↑	LPIPS↓
Multiply	Concat	31.042	0.968	0.039
Multiply	Sum	31.023	0.967	0.039
Sum	Multiply	30.585	0.965	0.044
Sum	Sum	25.227	0.928	0.090

Figure 5. **Synthesis Results with Various Fusion Designs.** HexPlane could give reasonable results with different fusion designs.

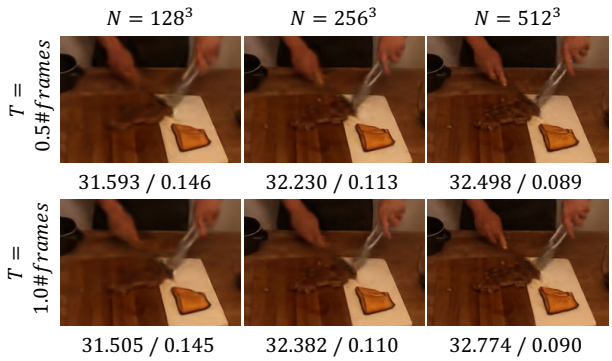


Figure 6. **Synthesis Results with Different Spacetime Grid Resolutions.** We show zoomed in synthesis results on Plenoptic Video dataset with space grid resolution ranging from 128^3 to 512^3 and time grid ranging from half to one of the video frame number. PSNR and LPIPS of the scene are reported below each images.

Could other fusion design work? In HexPlane, feature vectors are extracted from each plane and then fused into a single vector, which is multiplied by matrix $\mathbf{V}^{RF}$ later for final results. During fusion, features from paired planes are first element-wise multiplied (fusion one) and then concatenated into a single one (fusion two). We explore other fusion designs beyond this *Multiply-Concat* in Table 5.

Table 5 and Figure 5 reveal that *Multiply-Concat* is not the only feasible design. *Sum-Multiply* and its swapped version *Multiply-Sum* both provide good results although not optimal, indicating symmetry between multiplication and addition. *Sum-Sum* fails to give good results since it loses

Table 6. **Dynamic View Synthesis without MLPs.** HexPlane-SH is a pure explicit model without MLPs on D-NeRF dataset, which stores spherical harmonics (SH) as appearance features and directly regress RGB from it rather than MLPs. HexPlane-SH gives reasonable results and faster than HexPlane with MLP.

Model	PSNR↑	SSIM↑	LPIPS↓	Training Time↓
HexPlane	31.042	0.968	0.039	11m 30s
HexPlane-SH	29.284	0.952	0.056	10m 42s

the internal connections of paired planes. HexPlane is surprisingly robust and general to various fusion designs.

NDC Spherical NDC Spherical

Figure 7. **Synthesis Results at Extreme Views for NDC and Spherical Coordinates.** Scenes represented in NDC are assumed to be bounded along x, y axes, whose boundaries are observable at extreme views(top-left and top-right corners), leading to incorrect geometries and artifacts. Using spherical coordinate, our HexPlane could seamlessly represent dynamic unbounded scenes.

How does grid resolution effect results? We show qualitative results with various spacetime grid resolutions in Figure 6 and report its PSNR/LPIPS below zoomed-in images. Besides the space grid ranging from 128^3 to 512^3, we compare results with different time grid resolutions, ranging from half to the same as video frames. Higher resolutions of the space grid lead to better synthesis quality, shown by both images and metrics. HexPlane results are not noticeably affected by a smaller time grid resolution.

Could HexPlane work without MLP? We evaluate a pure explicit model in Table 6 without MLPs. Spherical harmonics (SH) [62] coefficients are computed from HexPlane, and RGB colors are calculated from SH along with view directions. Although it gives inferior results, this model has faster speeds since it is MLP-free. Further, it demonstrates HexPlane's ability to work with various color decodings.

4.3. Re-parameterization for Unbounded Scene

So far, HexPlane is restrictive to bounded scenes since its grid sampling fails to give reasonable results for out-of-boundary points, which is also a common issue for many explicit representations. Although normalized device coordinates (NDC) [32] could map points with infinite z into a bounded interval $[0, 1]$, it still requires bounded x, y values and face-forwarding assumptions. This limitation constrains the usage for real-world videos and leads to artifacts and incorrect geometries. Figure 7 shows synthesis results at extreme views for HexPlane using NDC with small x, y bounds, failing to model objects near or out of boundaries.

A natural way to convert unbounded scenes into bounded one is re-parameterizing (x, y, z, t) into (θ, ϕ, r, t), where $r = 1/\sqrt{x^2 + y^2 + z^2}$, θ, ϕ is the polar angle and azimuthal angle. During rendering, points are sampled with r linearly placed between 0 and 1.

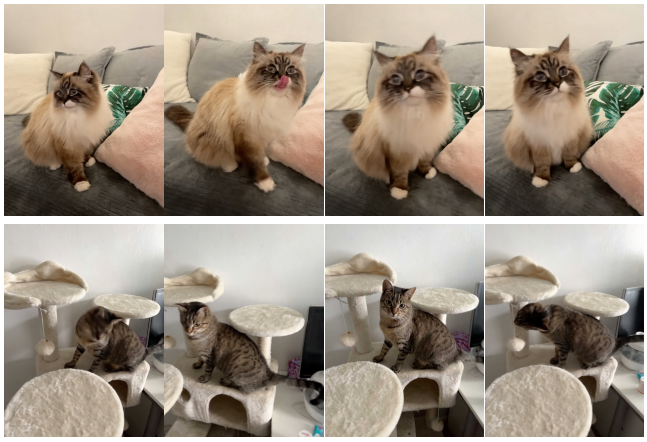


Figure 8. **Dynamic Novel View Synthesis on Videos Captured by iPhone.** We test HexPlane on casual videos captured by iPhone [14] and show synthesis results on novel times and views.

Could HexPlane work in spherical coordinate? Motivated by it, we construct HexPlane with θ, ρ, r, t axes without any special adjustments to validate this idea. Please note that our target is to validate whether HexPlane could work in spherical coordinates to model unbounded scenes instead of getting better numbers. This simple extension achieves 28.650 PSNR and 0.0966 LPIPS, which is still a good result, although not optimal. HexPlane is robust to work with spherical coordinates, supporting unbounded scenes.

4.4. View Synthesis Results on Real Captured Video

We test HexPlane with monocular videos captured by iPhone from [14], whose camera trajectories are relatively casual and closer to real-world use cases. We show synthesis results in 8. Without any deformation or category-specific priors, our method could give realistic synthesis results on these real-world monocular videos, faithfully modeling static backgrounds, casual motions of cats, typology changes (cat's tongue), and fine details like cat hairs.

5. Conclusion

We propose HexPlane, an explicit representation for dynamic 3D scenes, which represents dynamic scenes via six feature planes, and computes features of spacetime points via efficient sampling and fusion operations. Compared to implicit representation-based methods, it could achieve comparable or better synthesis quality for dynamic novel view synthesis, with hundreds of times accelerations.

In this paper, we aim to keep HexPlane neat and general, preventing introducing deformation, category-specific priors, or other specific tricks. Using these ideas to make

HexPlane better and faster would be an appealing future direction. Also, using HexPlane in other tasks except for dynamic novel view synthesis, e.g., spatial-temporal generation, would be interesting to explore. We hope HexPlane could contribute to a broad range of research in 3D.

Acknowledgments Toyota Research Institute provided funds to support this work but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. We thank Mohamed El Banani, Ziyang Chen, Linyi Jin and Shengyi Qian for helpful discussions.

References

[1] Chong Bao, Bangbang Yang, Zeng Junyi, Bao Hujun, Zhang Yinda, Cui Zhaopeng, and Zhang Guofeng. Neumesh: Learning disentangled neural mesh-based implicit field for geometry and texture editing. In *European Conference on Computer Vision (ECCV)*, 2022. 2

[2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 2

[3] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 2, 4

[4] Ang Cao, Chris Rockwell, and Justin Johnson. Fwd: Real-time novel view synthesis with forward warping and depth. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15713–15724, 2022. 2

[5] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16123–16133, 2022. 2, 3

[6] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021. 2

[7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *arXiv preprint arXiv:2203.09517*, 2022. 1, 2, 3, 5, 7

[8] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14124–14133, 2021. 2

[9] Yu Deng, Jiaolong Yang, Jianfeng Xiang, and Xin Tong. Gram: Generative radiance manifolds for 3d-aware image generation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10663–10673, 2022. 2

[10] Bernhard Egger, William AP Smith, Ayush Tewari, Stefanie Wuhrer, Michael Zollhoefer, Thabo Beeler, Florian Bernard, Timo Bolkart, Adam Kortylewski, Sami Romdhani, et al. 3d morphable face models—past, present, and future. *ACM Transactions on Graphics (TOG)*, 39(5):1–38, 2020. 12

[11] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. *arXiv preprint arXiv:2205.15285*, 2022. 2, 6, 12

[12] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 1, 2

[13] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021. 1, 2

[14] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. In *NeurIPS*, 2022. 8, 12

[15] 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. 2

[16] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 2

[17] Ajay Jain, Ben Mildenhall, Jonathan T. Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. 2022. 2

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 13

[19] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. *arXiv*, 2022. 2

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 13

[21] Ruilong Li, Julian Tanke, Minh Vo, Michael Zollhofer, Jurgen Gall, Angjoo Kanazawa, and Christoph Lassner. Tava: Template-free animatable volumetric actors. *ArXiv*, abs/2206.08929, 2022. 2

[22] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5521–5531, 2022. 1, 5, 6, 12, 13, 14, 15

[23] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6498–6508, 2021. 1, 2

[24] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *ArXiv*, abs/2007.11571, 2020. 2

[25] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *arXiv preprint arXiv:1906.07751*, 2019. 6

[26] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, Oct. 2015. 12

[27] Rafał K Mantiuk, Gyorgy Denes, Alexandre Chapiro, Anton Kaplanyan, Gizem Rufo, Romain Bachy, Trisha Lian, and Anjul Patney. Fovvideovdp: A visible difference predictor for wide field-of-view video. *ACM Transactions on Graphics (TOG)*, 40(4):1–19, 2021. 6, 13

[28] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021. 2

[29] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019. 2

[30] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P Srinivasan, and Jonathan T Barron. Nerf in the dark: High dynamic range view synthesis from noisy raw images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16190–16199, 2022. 2

[31] 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 Transactions on Graphics (TOG)*, 38(4):1–14, 2019. 6

[32] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2, 3, 5, 8, 13

[33] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 1, 2

[34] Michael Niemeyer, Jonathan T Barron, Ben Mildenhall, Mehdi SM Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5480–5490, 2022. 2, 4

[35] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2

[36] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020. 2

[37] Hao Ouyang, Bo Zhang, Pan Zhang, Hao Yang, Jiaolong Yang, Dong Chen, Qifeng Chen, and Fang Wen. Real-time neural character rendering with pose-guided multiplane images. *ArXiv*, abs/2204.11820, 2022. 2

[38] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021. 1, 2, 12, 19

[39] 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. 1, 2, 12

[40] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision*, pages 523–540. Springer, 2020. 2

[41] 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. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9050–9059, 2021. 2

[42] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv*, 2022. 2

[43] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021. 1, 2, 5, 6, 12, 13, 14, 16

[44] Shengyi Qian, Alexander Kirillov, Nikhila Ravi, Devendra Singh Chaplot, Justin Johnson, David F Fouhey, and Georgia Gkioxari. Recognizing scenes from novel viewpoints. *arXiv preprint arXiv:2112.01520*, 2021. 2

[45] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14315–14325, 2021. 2

[46] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33:20154–20166, 2020. 2

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 13

[48] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019. 2

[49] Vincent Sitzmann, Michael Zollhoefer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *ArXiv*, abs/1906.01618, 2019. 2

[50] Shih-Yang Su, Frank Yu, Michael Zollhoefer, and Helge Rhodin. A-nerf: Articulated neural radiance fields for learning human shape, appearance, and pose. In *NeurIPS*, 2021. 2

[51] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 1, 2

[52] Jiaming Sun, Xi Chen, Qianqian Wang, Zhengqi Li, Hadar Averbuch-Elor, Xiaowei Zhou, and Noah Snavely. Neural 3d reconstruction in the wild. *ACM SIGGRAPH 2022 Conference Proceedings*, 2022. 2

[53] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8238–8248, 2022. 2

[54] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. *arXiv preprint arXiv:2112.03907*, 2021. 2

[55] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3835–3844, 2022. 2

[56] 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 (CVPR)*, pages 13524–13534, June 2022. 2

[57] 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. 2

[58] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431, 2021. 2

[59] Hongyi Xu, Thiemo Alldieck, and Cristian Sminchisescu. H-nerf: Neural radiance fields for rendering and temporal reconstruction of humans in motion. In *NeurIPS*, 2021. 2

[60] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5438–5448, 2022. 2

[61] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Tsung-Yi Lin, Alberto Rodriguez, and Phillip Isola. NeRF-Supervision: Learning dense object descriptors from neural radiance fields. In *IEEE Conference on Robotics and Automation (ICRA)*, 2022. 2

[62] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *arXiv*, 2021. 1, 2, 5, 8

[63] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021. 2

[64] Jason Y. Zhang, Gengshan Yang, Shubham Tulsiani, and Deva Ramanan. Ners: Neural reflectance surfaces for sparse-view 3d reconstruction in the wild. In *NeurIPS*, 2021. 2

[65] Kai Zhang, Nicholas I. Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields. In *ECCV*, 2022. 2

[66] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 2, 12

[67] 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. 6

[68] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 13

[69] Xiaoshuai Zhang, Sai Bi, Kalyan Sunkavalli, Hao Su, and Zexiang Xu. Nerfusion: Fusing radiance fields for large-scale scene reconstruction. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5439–5448, 2022. 2

[70] Fuqiang Zhao, Wei Yang, Jiakai Zhang, Pei-Ying Lin, Yingliang Zhang, Jingyi Yu, and Lan Xu. Humannerf: Generalizable neural human radiance field from sparse inputs. *ArXiv*, abs/2112.02789, 2021. 2

[71] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew Davison. In-place scene labelling and understanding with implicit scene representation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 2

[72] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12786–12796, 2022. 2

A. Broader Impacts

Our work aims to design an explicit representation for dynamic 3D scenes. We only reconstruct existing scenes and render images from different viewpoints and timesteps. Therefore, we don't generate any new scenes or deceive contents which don't exist before. Our current method is not intended and can also not be used to create fake materials, which could mislead others.

We use *Plenoptic Video dataset* [22] in our experiments, which contains human faces in videos. This dataset is a public dataset with License: CC-BY-NC 4.0 with consent.

Our method is hundreds of times faster than existing methods, consuming significantly less computation resources. Considering the GPU resource usages, our method could save considerably carbon emission.

B. Limitations and Future Directions

For comprehensively understanding HexPlane, we discuss its limitations and potential future improvements.

The ultimate goal of our paper is to propose and validate an explicit representation for dynamic scenes instead of purchasing SOTA numbers. To this end, we intend to make HexPlanes simple and general, making minor assumptions about the scenes and not introducing complicated tricks to improve performance. This principle leads to elegant solutions while potentially limiting performance as well. In the following, we will discuss these in detail.

Many methods [11, 38, 39, 43] use deformation and canonical fields to represent dynamic 3D scenes with monocular videos, where spacetime points are mapped into a static 3D scene represented by a canonical field. Again, we don't employ deformation fields in our design since this assumption is not always held in the real world, especially for scenes with typology changes and new-emerging content. But this design is very effective for monocular videos since it introduces a solid information-sharing mechanism and allows learning 3D structures from very sparse views. HexPlane uses an inherent basis-sharing mechanism to cope with sparse observations. Although this design is shown to be powerful in our experiments, it is still less effective than the aforementioned deformation field, leading to degraded results for scenes with extremely sparse observations. Introducing deformation fields into HexPlane, like using HexPlane to represent deformation fields, would be an appealing improvement for monocular videos.

Similarly, category-specific priors like 3DMM [10] or SMPL [26] are even more powerful than deformation fields, which enormously improve results but are hardly limited to particular scenes. Combining these ideas with HexPlane for specific scenes would be very interesting.

Existing works demonstrated that explicit representations are prone to giving artifacts and require strong regularizations for good results, which also holds in HexPlane. There are color jittering and artifacts in the synthesized results, demanding stronger regularizations and other tricks to improve results further. Special spacetime regularizations and other losses like optical flow loss would be an interesting future direction to explore. Also, instead of simply representing everything using spherical coordinates in the paper, we could have a foreground and background model like NeRF++ [66], where the background is modeled in spherical coordinates. Having separate foreground background models could noticeably improve the results. Moreover, rather than using the same basis for representing a long video, using a different basis for different video clips may give better results. We believe HexPlane could be further improved with these adjustments.

Besides dynamic novel view synthesis, we believe HexPlane could be utilized in a broader range of research, like dynamic scene generation or edits.

C. Datasets and Baselines

C.1. License

We provide licenses of assets used in our paper.

Plenoptic Video Dataset [22]. We evaluate our method on all all public scenes of Plenoptic Video dataset [22], except a-synchronize scene "coffee-martini". The dataset is in https://github.com/facebookresearch/Neural_3D_Video with License CC-BY-NC 4.0

D-NeRF Dataset [43]. We use D-NeRF dataset provided in https://github.com/albertpumarola/D-NeRF.

iPhone Dataset [14]. The iPhone dataset is provided in https://github.com/KAIR-BAIR/dycheck/, licensed under Apache-2.0 license.

C.2. Baselines

Plenoptic Video Dataset Baselines [22]. For all baselines in this dataset, we use numbers reported in the original paper since these models are not publicly available.

D-NeRF Dataset Baselines [43]. D-NeRF model is in https://github.com/albertpumarola/D-NeRF and Tineuvox model [11] is in https://github.com/hustvl/TiNeuVox. Tineuvox is licensed under Apache License 2.0.

D. Training Details and More Results

D.1. Plenoptic Video Dataset [22].

Plenoptic Video Dataset [22] is a multi-view real-world video dataset, where each video is 10-second long. The training and testing views are shown in Figure 9.

We have $R_1 = 48, R_2 = 24, R_3 = 24$ for appearance HexPlance, where R_1, R_2, R_3 are basis numbers for

Figure 9. **Train and Test View of Plenoptic Video Dataset [22].** Plenoptic Video Dataset has 18 train views and 1 test view.

$XY-ZT, XZ-YT, YZ-XT$ planes. For opacity HexPlane, we set $R_1 = 24, R_2 = 12, R_3 = 12$. We have different R_1, R_2, R_3 since scenes in this dataset are almost face-forwarding, demanding better representation along the XY plane. The scene is modeled using *normalized device coordinate* (NDC) [32] with min boundaries $[-2.5, -2.0, 0.0]$ and max boundaries $[2.5, 2.0, 1.0]$.

Instead of giving the same grid resolutions along X, Y, Z axes, we adjust them based on their boundary distances. That is, we give larger grid resolution to axis ranging a longer distance, like X axis from -2.5 to 2.5, and provide smaller grid resolution to axis going a shorter length, like Z axis from 0 to 1. The ratio of grid resolutions for different axes is the same as their distance ratios, while the total grid size number is manually controlled.

During training, HexPlane starts with a space grid size of 64^3 and doubles its resolution at 70k, 140k, and 210k to 512^3. The emptiness voxel is calculated at 50k and 100k iterations. The learning rate for feature planes is 0.02, and the learning rate for $\mathbf{V}^{RF}$ and neural network is 0.001. All learning rates are exponentially decayed. We use Adam [18] for optimization with $\beta_1 = 0.9, \beta_2 = 0.99$. We apply Total Variational loss on all feature planes with $\lambda = 0.0001$.

We follow the hierarch training pipeline as [22]. *HexPlane* in Table 1 uses 650k iterations, with 300k stage one training, 250k stage two training and 100k stage three training. *HexPlane†* uses 350k iterations in total, with 100k stage one training, 150k stage two training and 100k stage three training. According to [22], stage one is a global-median-based weighted sampling with $\gamma = 0.001$; stage two is also a global-median-based weighted sampling with $\gamma = 0.02$; stage three is a temporal-difference-based weighted sampling with $\alpha = 0.1$.

In evaluation, D-SSIM is computed as $\frac{1-\text{MS-SSIM}}{2}$ and LPIPS [68] is calculated using AlexNet [20]. We use default settings for Just-Objectionable-Difference (JOD) [27].

Each scene results are in Table 7, and more visualizations are in Figure 10. We found that *HexPlane* gives visually more smooth results than *HexPlane†*. Since we don't have baseline results, we don't explore new evaluation metrics.

D.2. D-NeRF Dataset [43].

We have $R_1 = R_2 = R_3 = 48$ for appearance HexPlane since it has 360° videos. For opacity HexPlane, we set $R_1 = R_2 = R_3 = 24$. The bounding box has max boundaries $[1.5, 1.5, 1.5]$ and min boundaries $[-1.5, -1.5, -1.5]$.

During training, HexPlane starts with space grid size of 32^3 and upsamples its resolution at 3k, 6k, 9k to 200^3. The emptiness voxel is calculated at 4k and 10k iterations. Total training iteration is 25k. The learning rate for feature planes are 0.02, and learning rate for $\mathbf{V}^{RF}$ and neural network is 0.001. All learning rates are exponentially decayed. We use Adam [18] for optimization with $\beta_1 = 0.9, \beta_2 = 0.99$. During evaluation, the LPIPS is computed using VGG-Net [47] following previous works. We show per-scene quantitative results in Table 8 and visualizations in Figure 11.

D.3. Ablation Details.

For a fair comparison, we fix all settings in ablations.

Volume Basis represents 4D volumes as the weighted summation of a set of shared 3D volumes as Eq 2 in main paper, where each 3D volume is represented in Eq 1 format to save memory. The 3D volume $\mathbf{V}_t$ at time t is then:

$$\begin{aligned}\mathbf{V}_t &= \sum_{i=1}^{R_t} \mathbf{f}(t)_i \cdot \hat{\mathbf{V}}_i \\ &= \sum_{i=1}^{R_t} \mathbf{f}(t)_i (\sum_{r=1}^{R_1} \mathbf{M}_{r,i}^{XY} \circ \mathbf{v}_{r,i}^{Z} \circ \mathbf{v}_{r,i}^{1} + \sum_{r=1}^{R_2} \mathbf{M}_{r,i}^{XZ} \\ &\quad \circ \mathbf{v}_{r,i}^{Y} \circ \mathbf{v}_{r,i}^{2} + \sum_{r=1}^{R_3} \mathbf{M}_{r,i}^{YZ} \circ \mathbf{v}_{r,i}^{X} \circ \mathbf{v}_{r,i}^{3})\end{aligned} \tag{8}$$

Similarly, we use a piece-wise linear function to approximate $\mathbf{f}(t)$. In experiments, we set $R_1 = R_2 = R_3 = 16$ for appearance HexPlane and $R_1 = R_2 = R_3 = 8$ for opacity HexPlane. We evaluate $R_t = 8, 12, 16$ in experiments.

VM-T (Vector, Matrix and Time) uses Eq 3 in main paper to represent 4D volumes.

Table 7. **Results of Plenoptic Video Dataset [22].** We report results of each scene.

Model	Flame Salmon				Cook Spinach				Cut Roasted Beef			
	PSNR↑	D-SSIM ↓	LPIPS ↓	JOD ↑	PSNR↑	D-SSIM ↓	LPIPS ↓	JOD ↑	PSNR↑	D-SSIM ↓	LPIPS ↓	JOD ↑
HexPlane	29.470	0.018	0.078	8.16	32.042	0.015	0.082	8.32	32.545	0.013	0.080	8.59
HexPlane†	29.180	0.021	0.089	0.08	32.324	0.015	0.083	8.36	32.517	0.016	0.082	8.46
Model	Flame Steak				Sear Steak				Average			
	PSNR↑	D-SSIM ↓	LPIPS ↓	JOD ↑	PSNR↑	D-SSIM ↓	LPIPS ↓	JOD ↑	PSNR↑	D-SSIM ↓	LPIPS ↓	JOD ↑
HexPlane	32.080	0.011	0.066	8.61	32.387	0.011	0.070	8.66	31.705	0.014	0.075	8.47
HexPlane†	31.823	0.012	0.071	8.55	32.233	0.012	0.070	8.59	31.615	0.015	0.079	8.41

Table 8. **Per-Scene Results of D-NeRF Dataset [43].** We report results of each scene.

Model	Hell Warrior			Mutant			Hook		
	PSNR↑	SSIM↑	LPIPS ↑	PSNR↑	SSIM↑	LPIPS ↑	PSNR↑	SSIM↑	LPIPS ↑
T-NeRF	23.19	0.93	0.08	30.56	0.96	0.04	27.21	0.94	0.06
D-NeRF	25.02	0.95	0.06	31.29	0.97	0.02	29.25	0.96	0.11
TiNeuVox-S	27.00	0.95	0.09	31.09	0.96	0.05	29.30	0.95	0.07
TiNeuVox-B	28.17	0.97	0.07	33.61	0.98	0.03	31.45	0.97	0.05
HexPlane	24.24	0.94	0.07	33.79	0.98	0.03	28.71	0.96	0.05
Model	Bouncing Balls			Lego			T-Rex		
	PSNR↑	SSIM↑	LPIPS ↑	PSNR↑	SSIM↑	LPIPS ↑	PSNR↑	SSIM↑	LPIPS ↑
T-NeRF	37.81	0.98	0.12	23.82	0.90	0.15	30.19	0.96	0.13
D-NeRF	38.93	0.98	0.10	21.64	0.83	0.16	31.75	0.97	0.03
TiNeuVox-S	39.05	0.99	0.06	24.35	0.88	0.13	29.95	0.96	0.06
TiNeuVox-B	40.73	0.99	0.04	25.02	0.92	0.07	32.70	0.98	0.03
HexPlane	39.69	0.99	0.03	25.22	0.94	0.04	30.67	0.98	0.03
Model	Stand Up			Jumping Jacks			Average		
	PSNR↑	SSIM↑	LPIPS ↑	PSNR↑	SSIM↑	LPIPS ↑	PSNR↑	SSIM↑	LPIPS ↑
T-NeRF	31.24	0.97	0.02	32.01	0.97	0.03	29.51	0.95	0.08
D-NeRF	32.79	0.98	0.02	32.80	0.98	0.03	30.50	0.95	0.07
TiNeuVox-S	32.89	0.98	0.03	32.33	0.97	0.04	30.75	0.96	0.07
TiNeuVox-B	35.43	0.99	0.02	34.23	0.98	0.03	32.64	0.97	0.04
HexPlane	34.36	0.98	0.02	31.65	0.97	0.04	31.04	0.97	0.04

$$\mathbf{V}_t = \sum_{r=1}^{R_1} \mathbf{M}_r^{XY} \circ \mathbf{v}_r^Z \circ \mathbf{v}_r^1 \cdot \mathbf{f}_r^1(t) + \sum_{r=1}^{R_2} \mathbf{M}_r^{XZ} \circ \mathbf{v}_r^Y \circ \mathbf{v}_r^2 \cdot \mathbf{f}_r^2(t) + \sum_{r=1}^{R_3} \mathbf{M}_r^{ZY} \circ \mathbf{v}_r^X \circ \mathbf{v}_r^3 \cdot \mathbf{f}_r^3(t) \tag{9}$$

We evaluate $R_1 = R_2 = R_3 = 24, 48, 96$.

CP Decom. (CANDECOMP Decomposition) represents 4D volumes using a set of vectors for each axis.

$$\mathbf{V}_t = \sum_{r=1}^{R} \mathbf{v}_r^X \circ \mathbf{v}_r^Y \circ \mathbf{v}_r^Z \circ \mathbf{v}_r \cdot \mathbf{f}_r(t) \tag{10}$$

$\mathbf{v}^X, \mathbf{v}^Y, \mathbf{v}^Z$ are feature vectors corresponding to X, Y, Z axes. We evaluate $R = 48, 96, 192, 384$ in experiments.

D.4. Visualization of Feature Planes.

We visualize each channel of XT, ZT feature plane for opacity HexPlane in Figure 12. This HexPlane is trained in *Flame Salmon* scene in Plenoptic Video Dataset [22].

E. Failure Cases

HexPlane doesn't always give satisfactory results. It generates degraded results when objects move too fast or there are too few observations to synthesis details. Figure 13 shows failure cases and corresponding ground-truth images.

F. Failed Designs for Dynamic Scenes

Although HexPlane is a simple and elegant solution, it is not the instant solution we had for this task. In this section, we discuss other designs we tried. These designs could

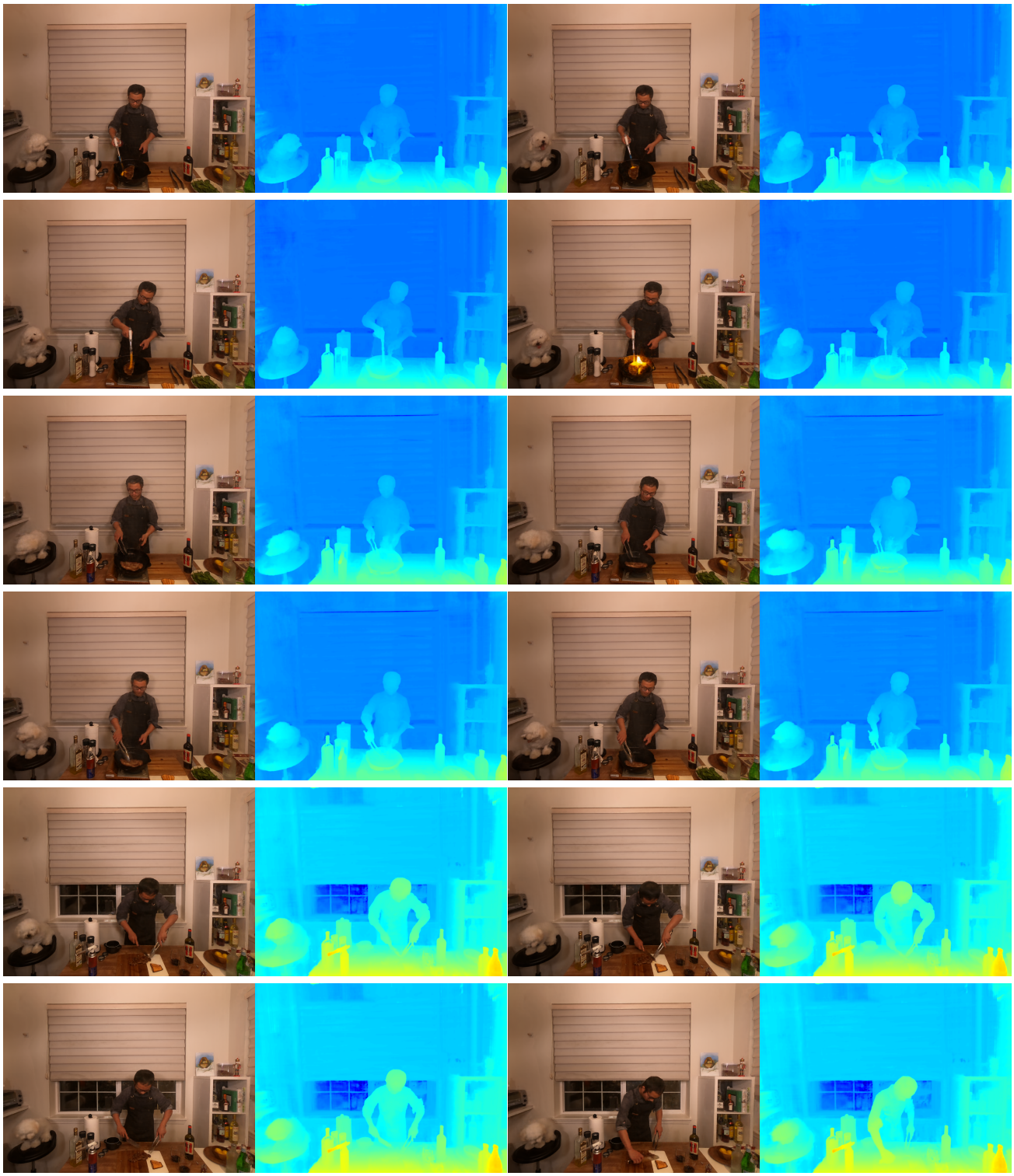


Figure 10. **View Synthesis Results and Depths at Test View on Plenoptic Video Dataset [22].**

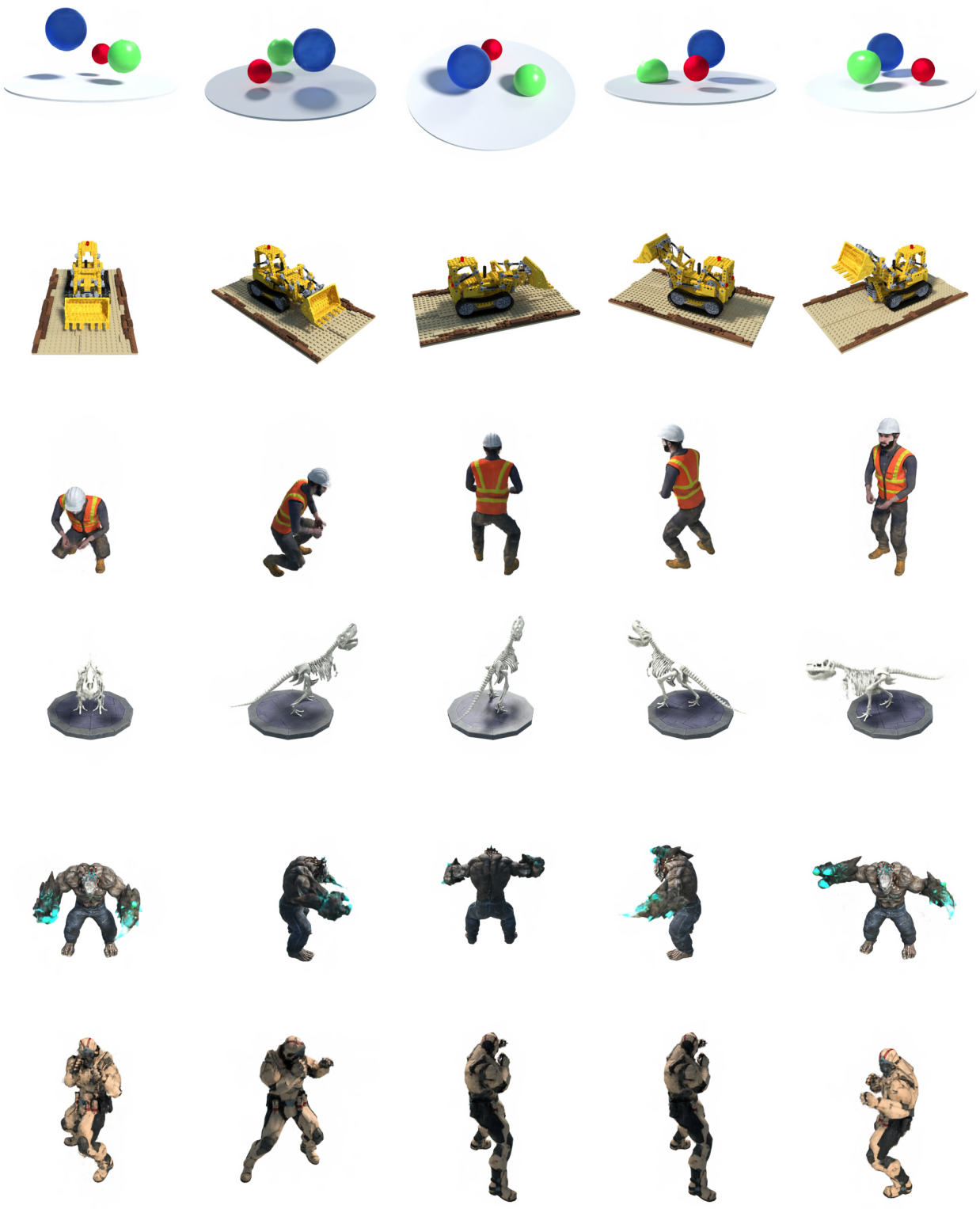
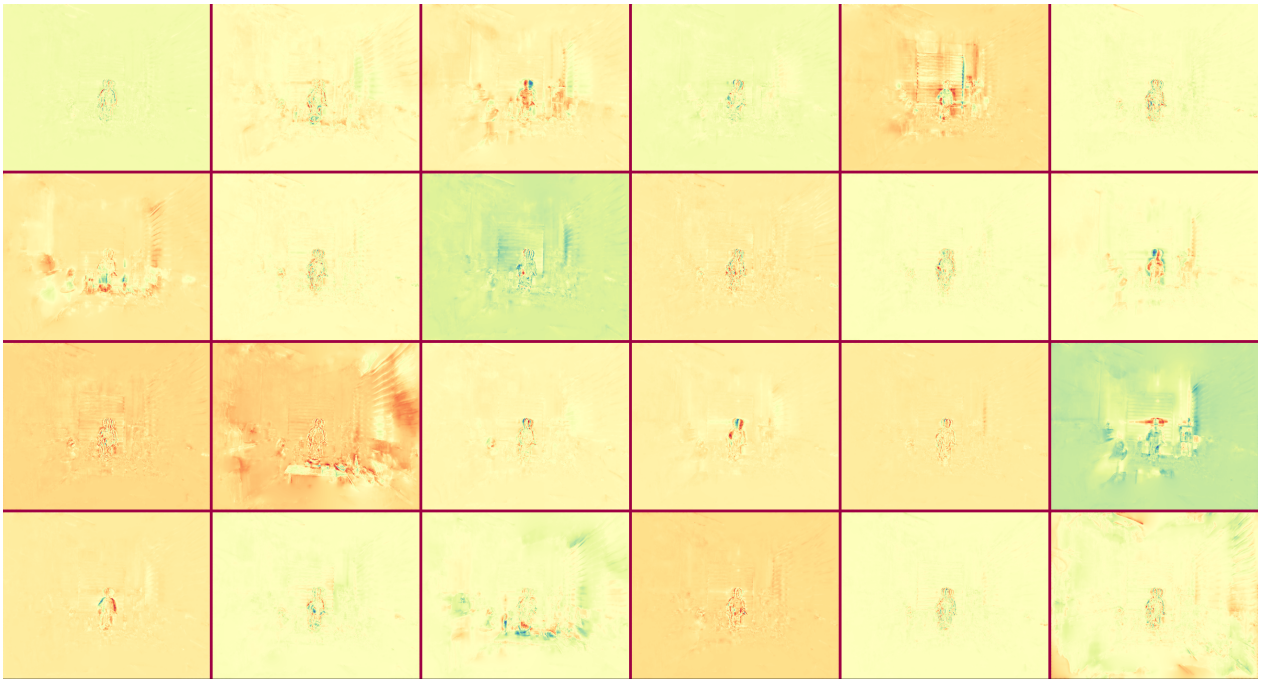
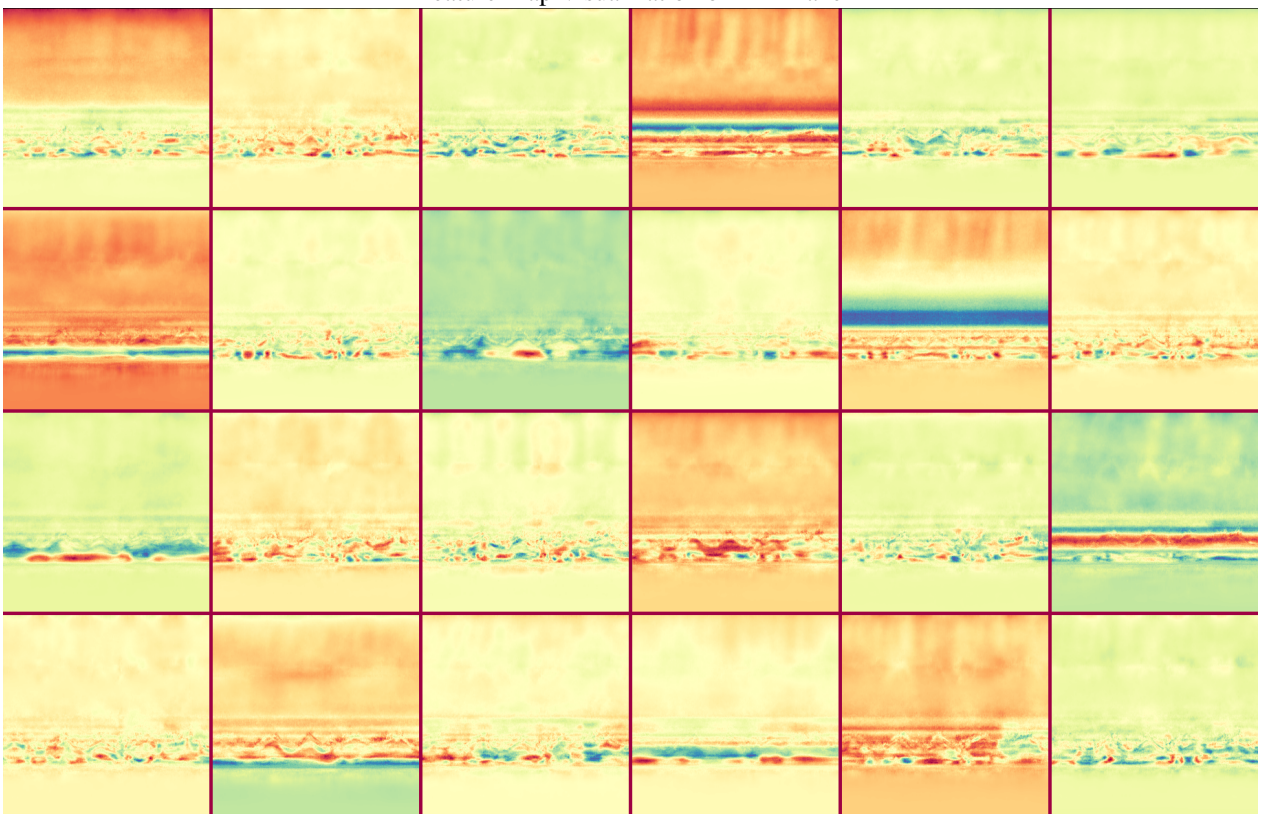


Figure 11. **View Synthesis Results on D-NeRF Dataset [43].**



Feature Map Visualization on XY Plane



Feature Map Visualization on ZT Plane

Figure 12. **Feature Map Visualization on *Flame Salmon* Scene.**

Figure 13. **Failure Cases from HexPlane.**

model the dynamic scenes while their qualities and speeds are not comparable to HexPlane. We discuss these "failed" designs, hoping they could inspire future work.

F.1. Fixed Basis for Time Axis

In Eq 2 of main paper, we use $\mathbf{f}(t)$ as the coefficients of basis volumes at time t. Its could be further expressed as:

$$\mathbf{V_t} = \sum_{i=1}^{R_t} \mathbf{f}(t)_i \cdot \hat{\mathbf{V}}_i = \hat{\mathbf{V}} \cdot \mathbf{f}(t) \tag{11}$$

where the second $\cdot$ is matrix-vector production; $\hat{\mathbf{V}} \in \mathbb{R}^{XYZFR_t}$ is the stack of $\{\hat{\mathbf{V}}_1, \ldots, \hat{\mathbf{V}}_{R_t}\}$; $\mathbf{f}(t) \in \mathbb{R}^{R_t}$ is a function of t. An interesting perspective to understand $\hat{\mathbf{V}}$ is: instead of storing static features with shape $\mathbb{R}^F$, every spatial point in 3D volume contains a feature matrix with shape $\mathbb{R}^{FR_t}$. And feature vectors at specific time t could be computed by inner product between $\mathbf{f}(t)$ and feature matrix. That is, $\mathbf{f}(t)$ is a set of basis functions w.r.t to time t and feature matrix contains coefficients of basis functions to approximate feature value changes along with time. Following the traditional approach of basis functions for time series, we use a set of sine/cosine functions as $\mathbf{f}(t)$.

While in practice, we found this implementation couldn't work since it requires enormous GPU memories. For instance, with $X = Y = Z = 128, R_t = 32, F = 27$, it uses 7GB to store such a representation and around 30GB during training because of back-propagation and keeping auxiliary terms of Adam. And it is extremely slow because of reading/writing values in memories.

Therefore, we apply tensor decomposition to reduce memory usages by factorizing volumes into matrixes $\mathbf{M}^{XY}, \mathbf{M}^{XZ}, \mathbf{M}^{YZ}$ and vectors $\mathbf{v}^X, \mathbf{v}^Y, \mathbf{v}^Z$ following Eq 1. Similarly, we add additional R_t dimension to matrixes $\mathbf{M}$ and vectors $\mathbf{v}$, leading to $\mathbf{MT}_r^{XY} \in \mathbb{R}^{XYR_T}, \mathbf{vT}_r^X \in \mathbb{R}^{XR_T}$. When calculating features from the representation,

$\mathbf{f}(t)$ is first multiplied with $\mathbf{MT}_i^{XY}$ and $\mathbf{vT}_i^X$ along the last dimension to get $\mathbf{M}$ and $\mathbf{v}$ at this time steps. We then calculate features using resulted $\mathbf{v}$ and $\mathbf{M}$ following Eq 1.

$\mathbf{f}(t)$ is designed to be like positional encoding, $\mathbf{f}(t) = [1, \sin(t), \cos(t), \sin(2*t), \cos(2*t), \sin(4*t), \cos(4*t), \cdots]$. We also try to use Legendre polynomials or Chebyshev polynomials to represent $\mathbf{f}(t)$. During training, we use weighted $L1$ loss to regularize $\mathbf{MT}_r, \mathbf{vT}_r$, and assign higher weights for high-frequency coefficients to keep results smooth. We also use the smoothly bandwidth annealing trick in [38] during training, which gradually introducing high-frequency components.

This design could model dynamic scenes while it suffers from severe color jittering and distortions. Compared to HexPlane, it has additional matrix-vector production, which reduces overall speeds.

F.2. Frequency-Domain Methods

We also tried another method from the frequency domain, which is orthogonal to the HexPlane idea. The notations of this section are slightly inconsistent with the notations of the main paper.

According to Fourier Theory, the value at (x, y, z, t) spacetime point could be represented in its frequency domain (we ignore feature dimension here for simplicity):

$$\mathbf{D}(x,y,z,t) = \sum_{u=1}^{U}\sum_{v=1}^{V}\sum_{w=1}^{W}\sum_{k=1}^{K} \widetilde{\mathbf{D}}(u,v,w,k) \cdot e^{-j2\pi(\frac{ux}{U}+\frac{vy}{V}+\frac{wz}{W}+\frac{kt}{K})} \tag{12}$$

$\widetilde{\mathbf{D}}$ is another 4D volume storing frequency weights, having the same size as $\mathbf{D}$. Storing $\widetilde{\mathbf{D}}$ is memory-consuming, and similarly, we apply tensor decomposition on this volume.

$$\begin{aligned}
&\mathbf{D}(x,y,z,t) = \sum_{u=1}^{U}\sum_{v=1}^{V}\sum_{w=1}^{W}\sum_{k=1}^{K}\sum_{r=1}^{R} \widetilde{\mathbf{v}}^U(u)_r \cdot \widetilde{\mathbf{v}}^V(v)_r \cdot \\
&\widetilde{\mathbf{v}}^W(w)_r \cdot \widetilde{\mathbf{v}}^K(k)_r \cdot e^{-j2\pi(\frac{ux}{U}+\frac{vy}{V}+\frac{wz}{W}+\frac{kt}{K})} \\
&= \sum_{u=1}^{U}\sum_{v=1}^{V}\sum_{w=1}^{W}\sum_{k=1}^{K}\sum_{r=1}^{R} (\widetilde{\mathbf{v}}^U(u)_r \cdot e^{-j2\pi\frac{ux}{U}})(\widetilde{\mathbf{v}}^V(v)_r \cdot e^{-j2\pi\frac{vy}{V}}) \\
&\cdot (\widetilde{\mathbf{v}}^W(w)_r e^{-j2\pi\frac{wz}{W}}) \cdot (\widetilde{\mathbf{v}}^K(k)_r e^{-j2\pi\frac{kt}{K}})) \\
&= \sum_{r=1}^{R} (\sum_{u=1}^{U} \widetilde{\mathbf{v}}^U(u)_r \cdot e^{-j2\pi\frac{ux}{U}}) \cdot \sum_{v=1}^{V} (\widetilde{\mathbf{v}}^V(v)_r \cdot e^{-j2\pi\frac{vy}{Y}}) \cdot \\
&(\sum_{w=1}^{W} \widetilde{\mathbf{v}}^W(w)_r e^{-j2\pi\frac{wz}{W}}) \cdot (\sum_{k=1}^{K} \widetilde{\mathbf{v}}^K(k)_r e^{-j2\pi\frac{kt}{K}})
\end{aligned} \tag{13}$$

where $\widetilde{\mathbf{v}}^U, \widetilde{\mathbf{v}}^V, \widetilde{\mathbf{v}}^W, \widetilde{\mathbf{v}}^K$ are the decomposed vectors from $\widetilde{\mathbf{D}}$ along U, V, W, K axes using CANDECOMP Decomposition, which axes are related to x, y, z, t in time domain. Instead of storing the 4D frequency volume and computing values by traversing all elements inside this volume using Eq 5, we decompose 4D volumes into many single vectors and calculate values by summation along each axis, significantly reducing computations.

Similarly, we apply weight $L1$ and smoothly bandwidth annealing trick on vector weights. We also try wavelet series instead of Fourier series, and other decompositions. We found this method leads to less-saturated colors and degraded details, which is shown in videos.

Also, this method replaces grid sampling of HexPlane by inner product, which is less efficient and leads to slow speeds.