HexPlane: A Fast Representation for Dynamic Scenes

Ang Cao  Justin Johnson
University of Michigan, Ann Arbor
{ancao, justincj}@umich.edu

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. HexPlane is a simple and effective solution for representing 4D volumes, and we hope they can broadly contribute to modeling spacetime for dynamic 3D scenes.

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.

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 in 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 [42]. 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 [28, 29] or a deformation to a canonical static scene [16, 49, 50, 54]. 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.

Several recent methods for modeling static scenes have demonstrated tremendous speedups over NeRF through the use of explicit and hybrid methods [7, 43, 66, 81]. These methods use an explicit spatial data structure that stores explicit scene data [14, 81] or features that are decoded by a tiny MLP [7, 43, 66]. This decouples a model’s capacity from its speed, and allows high-quality images to be rendered in realtime [43]. While effective, these methods have thus far been applied only to static scenes.

In this paper, we aim to design an explicit representation of 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; naively 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.

1Project page: https://caoang327.github.io/HexPlane.
We overcome these challenges with our novel HexPlane architecture. Inspired by factored representations for static scenes [5, 7, 51], 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 [42].

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 [28] we match the image quality of prior work but improve training time by $>100x$; we also outperform prior approaches on a monocular video dataset [54]. 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 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 HexPlane will be useful for a broad range of research in dynamic scenes [61].

2. Related Work

Neural Scene Representations. Using neural networks to implicitly represent 3D scenes [39, 46, 62, 63, 67, 75] has achieved exciting progress recently. NeRF [42] and its variants [2, 3, 40, 44, 69, 71, 80, 87] show impressive results on novel view synthesis [9, 75, 82, 94] and many other applications including 3D reconstruction [38, 67, 85, 89, 95], semantic segmentation [25, 55, 93], generative model [5, 6, 10, 45, 58, 77], and 3D content creation [1, 22, 30, 48, 53, 72, 86].

Implicit neural representations exhibit remarkable rendering quality, but they suffer from slow rendering speeds due to the numerous costly MLP evaluations required for each pixel. To address this challenge, many recent papers propose hybrid representations that combine a fast explicit scene representation with learnable neural network components, providing significant speedups over purely implicit methods. Various explicit representations have been investigated, including sparse voxels [14, 34, 59, 66], low-rank components [5, 7, 31, 51], point clouds [4, 21, 79, 92, 96] and others [8, 36, 43, 68, 90]. However, these approaches assume static 3D scenes, leaving explicit representations for dynamic scenes unexplored. This paper provides an explicit model for dynamic scenes, substantially 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, enabling numerous real-world applications [27, 47, 52, 65, 78, 84, 91]. One line of research represents dynamic scenes by extending NeRF with an additional time dimension (T-NeRF) or additional latent code [16, 28, 29, 76]. Despite the ability to represent general typology changes, they suffer from a severely under-constrained problem, requiring additional supervision like depths, optical flows or dense observations for decent results. Another line of research employs individual MLPs to represent a deformation field and a canonical field [11, 49, 50, 54, 70, 83], where the canonical field depicts a static scene, and the deformation field learns coordinate maps to the canonical space over time. We propose a simple yet elegant solution for dynamic scene representation using six feature planes, making minimal assumptions about the underlying scene.

Recently, MAV3D [61] adopted our design for text-to-4D dynamic scene generation, demonstrating an exciting direction for dynamic scenes beyond reconstruction.

Accelerating NeRFs. Many works have been proposed to accelerate NeRF at diverse stages. Some methods improve inference speeds of trained NeRFs by optimizing the computation [18, 20, 56, 81]. Others reduce the training times by learning a generalizable model [9, 24, 74, 75]. Recently, rendering speeds during both stages are substantially reduced by using explicit-implicit representations [5, 7, 14, 33, 43, 66]. In line with this idea, we propose an explicit representation for dynamic fields to accelerate dynamic NeRFs.

Very recently, several concurrent works have aimed to accelerate dynamic NeRFs. [12, 15, 19, 32, 73] use time-aware MLPs to regress spacetime points’ colors or deformations from canonical spaces. However, they remain partially implicit for dynamic fields, as they rely on MLPs with time input to obtain spacetime features. In contrast, our paper proposes a more elegant and efficient explicit representation for dynamic fields without using time-aware MLPs. Like [26], NeRFPlayer [64] uses a highly compact 3D grid at each time step for 4D field representation, which results in substantial memory costs for lengthy videos.

Tensor4D [60] shares a similar idea as ours, which represents dynamic scenes with 9 planes and multiple MLPs. D-NeRF [23] regards dynamic fields as 5D tensors and applies CP/MM decomposition on them for compact representation. Our paper is most closely related to $K$-Planes [13], which also employs six feature planes for representation.
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 [42], 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 target 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 under-explored. 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 \( D \) comprising independent static 3D volumes per time step \( \{V_1, V_2, \ldots, V_T\} \). 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 \( V \in \mathbb{R}^{XY,ZT} \) as a sum of vector-matrix outer products:

\[
V = \sum_{r=1}^{R_1} M_{X}^{XY} \odot v_r^{XY} \odot v_r^X + \sum_{r=1}^{R_2} M_{Z}^{XY} \odot v_r^Y \odot v_r^Z + \sum_{r=1}^{R_3} M_{Z}^{XY} \odot v_r^X \odot v_r^3
\]

where \( \odot \) is outer product; \( M_{X}^{XY} \odot v_r^Z \odot v_r^1 \) is a low-rank component of \( V \); \( M_{X}^{XY} \in \mathbb{R}^{XY} \) is a matrix spanning the \( X \) and \( Y \) axes, and \( v_r^Z \in \mathbb{R}^Z, v_r^1 \in \mathbb{R}^F \) are vectors along the \( Z \) and \( F \) axes. \( R_1, R_2, R_3 \) are the number of low-rank components. With \( R = R_1 + R_2 + R_3 \ll N \), 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 \( \hat{V}_t \) at time \( t \) as the weighted sum of a set of shared 3D basis volumes \( \{V_1, \ldots, V_{R_t}\} \); then

\[
\hat{V}_t = \sum_{i=1}^{R_t} f(t)_i \cdot \hat{V}_i
\]

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 \( f(t) \in \mathbb{R}^{R_t} \) gives weights for the shared volumes as a function of \( t \). Shared volumes allow information to be shared across time. In practice each \( V_t \) 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 low-rank components across all shared volumes. The 3D volume \( V_t \) at time \( t \) is then

\[
V_t = \sum_{r=1}^{R_t} \mathbf{M}^{XY}_{r} \odot \mathbf{v}^{Z}_{r} \odot \mathbf{v}^{1}_{r} \cdot f^1(t)_r + \sum_{r=1}^{R_3} \mathbf{M}^{XZ}_{r} \odot \mathbf{v}^{Y}_{r} \odot \mathbf{v}^{2}_{r} \cdot f^2(t)_r + \sum_{r=1}^{R_3} \mathbf{M}^{Yz}_{r} \odot \mathbf{v}^{X}_{r} \odot \mathbf{v}^{3}_{r} \cdot f^3(t)_r \tag{3}
\]

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

In this formulation, \( f^i(t) \) captures the model’s dependence on time. The correct mathematical form for \( f^i(t) \) is not obvious. We initially framed \( f^i(t) \) 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. \( f^i(t) \) could be an arbitrary nonlinear mapping, represented as an MLP; however this would be slow. As a pragmatic trade-off between flexibility and speed, we represent \( f^i(t) \) 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: \( f^i(t) \) 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 \( \mathbf{v}^Z_{r} \cdot f^1(t)_r \) 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:

\[
D = \sum_{r=1}^{R_1} \mathbf{M}^{XY}_{r} \odot \mathbf{M}^{ZT}_{r} \odot \mathbf{v}^{1}_{r} + \sum_{r=1}^{R_3} \mathbf{M}^{XZ}_{r} \odot \mathbf{M}^{YT}_{r} \odot \mathbf{v}^{2}_{r} + \sum_{r=1}^{R_3} \mathbf{M}^{YZ}_{r} \odot \mathbf{M}^{XT}_{r} \odot \mathbf{v}^{3}_{r} \tag{4}
\]

where each \( \mathbf{M}^{AB}_{r} \in \mathbb{R}^{AB} \) 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 \( D \) which maps a point \( (x, y, z, t) \) to an \( F \)-dimensional feature:

\[
D(x, y, z, t) = \langle \mathbf{P}_{x,y}^{XYR_1} \odot \mathbf{P}_{z}^{ZTR_1} \rangle \mathbf{V}_R \mathbf{F} + \langle \mathbf{P}_{x,z}^{XYR_2} \odot \mathbf{P}_{y}^{YTR_2} \rangle \mathbf{V}_R \mathbf{F} + \langle \mathbf{P}_{y,z}^{YZR_3} \odot \mathbf{P}_{x}^{XTR_3} \rangle \mathbf{V}_R \mathbf{F} \tag{5}
\]

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 \( \mathbf{M}^{XY} \) to a 3D tensor, and \( \mathbf{V}_R \mathbf{F} \) stacks all \( \mathbf{v}^{1} \) 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 \( \mathbf{V}_R \mathbf{F} \) into \( \mathbf{V}_R \mathbf{F} \) and rewrite Eq 5 as:

\[
\begin{align*}
\langle \mathbf{P}_{x,y}^{XYR_1} \odot \mathbf{P}_{z}^{ZTR_1} \rangle \mathbf{V}_R \mathbf{F} + \langle \mathbf{P}_{x,z}^{XYR_2} \odot \mathbf{P}_{y}^{YTR_2} \rangle \mathbf{V}_R \mathbf{F} + \langle \mathbf{P}_{y,z}^{YZR_3} \odot \mathbf{P}_{x}^{XTR_3} \rangle \mathbf{V}_R \mathbf{F}
\end{align*}
\]

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 queried from HexPlane, and the final RGB color is regressed from a tiny MLP with aperture and appearance feature are queried 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:

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

\( \mathcal{L}_{\text{reg}}, \lambda_{\text{reg}} \) are regularization and its weight; \( \mathcal{R} \) is the set of rays and \( \hat{C}(r), C(r) \) are rendered and GT colors of ray \( r \).

**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 [44] to reduce artifacts.

**Coarse to Fine Training.** A coarse-to-fine scheme is also employed like [7, 81], where the resolution of grids gradually grows during training. This design accelerates the training and provides an implicit regularization on nearby grids.
Figure 3. **High-Quality Dynamic Novel View Synthesis on Plenoptic Video dataset** [28]. 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.

**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.

4. **Experiments**

We evaluate HexPlane, our proposed explicit representation, on dynamic novel view synthesis tasks with challenging datasets, comparing its performance and speed to state-of-the-art methods. Through extensive ablation studies, we explore its advantages and demonstrate its robustness to different feature fusion mechanisms, coordinate systems, and decoding mechanisms. As our objective is to demonstrate the effectiveness of this simple design, we prioritize HexPlane’s simplicity and generality without implementing intricate tricks for performance enhancement.

4.1. **Dynamic Novel View Synthesis Results**

For a comprehensive evaluation, we use two datasets with distinct settings: the high-resolution, multi-camera *Plenoptic Video dataset* [28], with challenging dynamic content and intricate visual details; the monocular *D-NeRF dataset* [54], featuring synthetic objects. *Plenoptic Video dataset* assesses HexPlane’s representational capacity for long videos with complex motions and fine details, while *D-NeRF dataset* tests its ability to handle monocular videos and extremely sparse observations (with teleporting [17]).

*Plenoptic Video dataset* [28] is a real-world dataset captured by a multi-view camera system using 21 GoPro at 2028 × 2704 (2.7K) resolution and 30 FPS. Each scene comprises 19 synchronized, 10-second videos, with 18 designated for training and one for evaluation. This dataset is suitable to test the representation ability as it features complex and challenging dynamic content such as highly specular, translucent, and transparent objects; topology changes; moving self-casting shadows; fire flames and strong viewpoint-dependent effects for moving objects; and so on.

For a fair comparison, we adhere to the same training and evaluation pipelines as DyNeRF [28] with slight changes due to GPU resources. [28] 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 a 4096 batch size and the same iteration numbers, which is 6× fewer sampling. We follow the same importance sampling design and hierarch training as [28], with 512 spatial grid sizes and 300 time grid sizes. The scene is in NDC [42].

As shown in Figure 3, HexPlane delivers high-quality dynamic novel view synthesis across various times and viewpoints. It accurately models real-world scenes with intricate motions and challenging visual features, such as flames, showcasing its robust representational capabilities.

Quantitative comparisons with SOTA methods are in Table 1, with baseline results from [28] paper. PSNR, struc-
Table 1. Quantitative Comparisons on Plenoptic Video dataset [28]. We report synthesis quality, training times (measured in GPU hours) with speedsups relative to DyNeRF [28]. With 672× speedsups, 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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Steps</th>
<th>PSNR↑</th>
<th>D-SSIM↓</th>
<th>LPIPS↓</th>
<th>JOD↑</th>
<th>Training Time↓</th>
<th>Speeds-up ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Volumes [35]</td>
<td>-</td>
<td>22.800</td>
<td>0.062</td>
<td>0.295</td>
<td>6.50</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LLFF [41]</td>
<td>-</td>
<td>23.239</td>
<td>0.076</td>
<td>0.235</td>
<td>6.48</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NeRF-T [28]</td>
<td>-</td>
<td>28.449</td>
<td>0.023</td>
<td>0.100</td>
<td>7.73</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DyNeRF [28]</td>
<td>650k</td>
<td>29.581</td>
<td>0.020</td>
<td>0.099</td>
<td>8.07</td>
<td>1344h</td>
<td>1×</td>
</tr>
<tr>
<td>HexPlane</td>
<td>650k</td>
<td>29.470</td>
<td>0.018</td>
<td>0.078</td>
<td>8.16</td>
<td>12h</td>
<td>112×</td>
</tr>
<tr>
<td>HexPlane†</td>
<td>100k</td>
<td>29.263</td>
<td>0.020</td>
<td>0.097</td>
<td>8.14</td>
<td>2h</td>
<td>672 ×</td>
</tr>
<tr>
<td>HexPlane-all</td>
<td>650k</td>
<td>31.705</td>
<td>0.014</td>
<td>0.075</td>
<td>8.47</td>
<td>12h</td>
<td>112 ×</td>
</tr>
<tr>
<td>HexPlane-all†</td>
<td>100k</td>
<td>31.569</td>
<td>0.016</td>
<td>0.089</td>
<td>8.36</td>
<td>2h</td>
<td>672 ×</td>
</tr>
</tbody>
</table>

Table 2. Quantitative Results on D-NeRF dataset [54]. Without deformation, HexPlane has comparable or better results compared to other deformation-based methods, and is noticeably faster.

<table>
<thead>
<tr>
<th>Model</th>
<th>Deform.</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
<th>Training Time↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-NeRF [54]</td>
<td>29.51</td>
<td>0.95</td>
<td>0.08</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D-NeRF [54]</td>
<td>✓</td>
<td>30.50</td>
<td>0.95</td>
<td>0.07</td>
<td>20 hours</td>
</tr>
<tr>
<td>Tineuvox-S [12]</td>
<td>✓</td>
<td>30.75</td>
<td>0.96</td>
<td>0.07</td>
<td>12n 10s</td>
</tr>
<tr>
<td>Tineuvox-B [12]</td>
<td>✓</td>
<td>32.67</td>
<td>0.97</td>
<td>0.04</td>
<td>49m 46s</td>
</tr>
<tr>
<td>HexPlane (ours)</td>
<td>31.04</td>
<td>0.97</td>
<td>0.04</td>
<td>11m 30s</td>
<td></td>
</tr>
</tbody>
</table>

We run deep introspections to HexPlane by answering questions with extensive ablations. Ablations are conducted mainly on D-NeRF [54] dataset because of efficiency.

4.2. Ablations and Analysis

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 \( R \): (1). Volume Basis represents 4D volumes as weighted summation of a set of shared 3D volumes as Eq 2, which 3D volume is represented as Eq 1; (2). \( VM-T \) (vector, matrix and time) uses Eq 3 representing 4D volumes;
Table 3. **Quantitative Results for Different Factorizations.** Various factorization designs are evaluated on D-NeRF dataset with different \( R \) (basis number). HexPlane achieves the best quality and speed among all methods.

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

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

<table>
<thead>
<tr>
<th>Model</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
<th>Training Time↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial Planes</td>
<td>20.369</td>
<td>0.879</td>
<td>0.148</td>
<td>9m 02s</td>
</tr>
<tr>
<td>Spatial-Temporal Planes</td>
<td>21.112</td>
<td>0.879</td>
<td>0.148</td>
<td>9m 29s</td>
</tr>
<tr>
<td>DoublePlane (XY-ZT)</td>
<td>30.370</td>
<td>0.961</td>
<td>0.054</td>
<td>8m 04s</td>
</tr>
<tr>
<td>HexPlane-Swap</td>
<td>28.562</td>
<td>0.954</td>
<td>0.065</td>
<td>11m 44s</td>
</tr>
<tr>
<td>HexPlane</td>
<td>31.042</td>
<td>0.968</td>
<td>0.039</td>
<td>11m 30s</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Fusion-One</th>
<th>Fusion-Two</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concat</td>
<td>31.042</td>
<td>0.968</td>
<td>0.039</td>
<td></td>
</tr>
<tr>
<td>Multiply</td>
<td>30.345</td>
<td>0.966</td>
<td>0.041</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>25.428</td>
<td>0.931</td>
<td>0.084</td>
<td></td>
</tr>
<tr>
<td>Multiply</td>
<td>30.585</td>
<td>0.965</td>
<td>0.044</td>
<td></td>
</tr>
</tbody>
</table>

(3). **CP Decom.** (CANDECOMP Decomposition) follows [7], which represents 4D volumes using a set of vectors for each axis. Implementation details are shown 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 a 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 \( R \) may lead to the 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: \( P^{XY}, P^{XZ}, P^{YZ} \), and **Spatial-Temporal Planes** contain the left three spatial-temporal planes; **DoublePlane** contains only one group of paired planes, i.e. \( P^{XY}, P^{ZT} \); **HexPlane-Swap** groups planes with repeated axes like \( P^{XY}, P^{XT} \). 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 only one group of paired planes, i.e. \( P^{XY}, P^{ZT} \) and then concatenated into a single one (fusion one). We explore other fusion designs beyond this Multiply-Concat.

Table 5 shows that Multiply-Concat is not the sole viable design. **Sum-Multiply** and swapped counterpart Multiply-Concat...
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 regresses RGB from it rather than MLPs. HexPlane-SH gives reasonable results and faster than HexPlane with MLP.

<table>
<thead>
<tr>
<th>Model</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
<th>Training Time↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>HexPlane</td>
<td>31.042</td>
<td>0.968</td>
<td>0.039</td>
<td>11m 30s</td>
</tr>
<tr>
<td>HexPlane-SH</td>
<td>29.284</td>
<td>0.952</td>
<td>0.056</td>
<td>10m 42s</td>
</tr>
</tbody>
</table>

Figure 6. 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. Sum both yield good results, albeit not optimal, highlighting an intriguing symmetry between multiplication and addition. Multiply-Multiply also produces satisfactory outcomes, while Sum-Sum or Sum-Concat fail, illustrating the capacity limitations of addition compared to multiplication. Overall, HexPlane is remarkably robust to various fusion designs. We show complete results and analysis in Supp.

Spherical Harmonics Color Decoding. Instead of regressing colors from MLPs, we evaluate a pure explicit model in Table 6 without MLPs. Spherical harmonics (SH) coefficients are computed directly from HexPlanes, and decoded to RGBs with view directions. Using SH allows faster rendering speeds at a slightly reduced quality. We find that optimizing SH for dynamic scenes is more challenging compared to [7, 14], which is an interesting future direction.

Spherical Coordinate for Unbounded Scenes. HexPlane is limited to bounded scenes because grid sampling fails for out-of-boundary points, which is a common issue among explicit representations. Even normalized device coordinates (NDC) [42] still require bounded $x$, $y$ values and face-forwarding assumptions. This limitation constrains the usage for real-world videos, leading to artifacts and incorrect geometries as shown in Figure 6.

To address it, we re-parameterize $(x, y, z, t)$ into spherical coordinate $(\theta, \phi, r, t)$ and build HexPlane with $\theta, \rho, r, t$ axes, where $r = \frac{1}{\sqrt{x^2 + y^2 + z^2}}$, $\theta, \phi$ is the polar angle and azimuthal angle. During rendering, points are sampled with $r$ linearly placed between 0 and 1. Without any special adjustments, HexPlane can represent dynamic fields with spherical coordinates, and deliver satisfactory results, which provides a solution for modeling unbounded scenes and exhibits robustness to different coordinate systems.

Figure 7. Dynamic Novel View Synthesis on Videos Captured by iPhone. We test HexPlane on casual videos captured by iPhone [17] and show synthesis results across novel timesteps and views. Row one are results with interpolated camera poses, while Row two shows results with extrapolated viewpoints, which are significantly distinct from camera poses used for video captures.

4.4. View Synthesis Results on Real Captured Video

We test HexPlane with monocular videos captured by iPhone from [17], whose camera trajectories are relatively casual and closer to real-world use cases. We show synthesis results in 7. 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 using six feature planes, which computes features of spacetime points via sampling and fusions. Compared to implicit representations, it could achieve comparable or even better synthesis quality for dynamic novel view synthesis, with over 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 [61]. 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 Shengyi Qian for the title suggestion, David Fouhey, Mohamed El Banani, Ziyang Chen, Linyi Jin and for helpful discussions and feedbacks.

---

2Further tuning of initialization/other factors may lead to better results.
References


Tava: Template-free animatable volumetric actors. *ArXiv*, abs/2206.08929, 2022. 2


[51] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy


[76] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint


