Streaming Radiance Fields for 3D Video Synthesis

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present an explicit-grid based method for efficiently reconstructing streaming radiance fields for novel view synthesis of real world dynamic scenes. Instead of training a single model that combines all the frames, we formulate the dynamic modeling problem with an incremental learning paradigm in which per-frame model difference is trained to complement the adaption of a base model on the current frame. By exploiting the simple yet effective strategy of tuning with narrow bands, the proposed method achieves a feasible framework for handling on-the-fly video sequences with high training efficiency. The storage overhead induced by using explicit grid can be significantly reduced by using model difference based compression. We also introduce a curriculum learning strategy to further accelerate model optimization for each frame. Experiments on challenging video sequences demonstrate that our approach is capable of achieving a training speed of 10 seconds per-frame with competitive rendering quality, which attains $1500\times$ speedup over the state-of-the-art implicit methods.

1 Introduction

3D video synthesis aims to realize free-viewpoint photo-realistic rendering for dynamic scenes, which are typically recorded via a set of cameras (with known poses) from multiple views, and has attracted much research effort due to its potential value in a wide range of applications in VR/AR. Traditional techniques like mesh [1, 2], multi-sphere image [3], and depth [4] estimation are usually used for modeling and representing dynamic scenes. However, the arbitrary geometry and complex appearance exhibited in real-world scenarios pose challenges for using a general methodology to pursue high-performance modeling.

Neural radiance fields (NeRF) [5, 6] have recently emerged as a new methodology for effectively reconstructing and rendering static scenes via neural networks. By learning a continuous mapping between 3D points (given view directions) and its corresponding RGB radiance color and opacity equipped with volumetric rendering techniques [7], these methods have shown compelling ability for rendering high-fidelity images, while encountering the bottleneck of costly training and inference as relying on a tremendous amount of computations through networks. When extending the paradigm to a dynamic scene, time steps are embedded as additional input [8] to achieve time-dependent rendering. However, the training overhead linearly increases with time, leading to enormous training costs when dealing with long sequences (e.g., costing about 56 GPU days for training 300 frames). The implicit formulation essentially suiting offline processing may not be easily generalized to handling online video sequences.

In this work, we propose a novel method for reconstructing streaming radiance fields for novel view synthesis of 3D videos. We formalize the modeling of a dynamic scene in an incremental learning

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

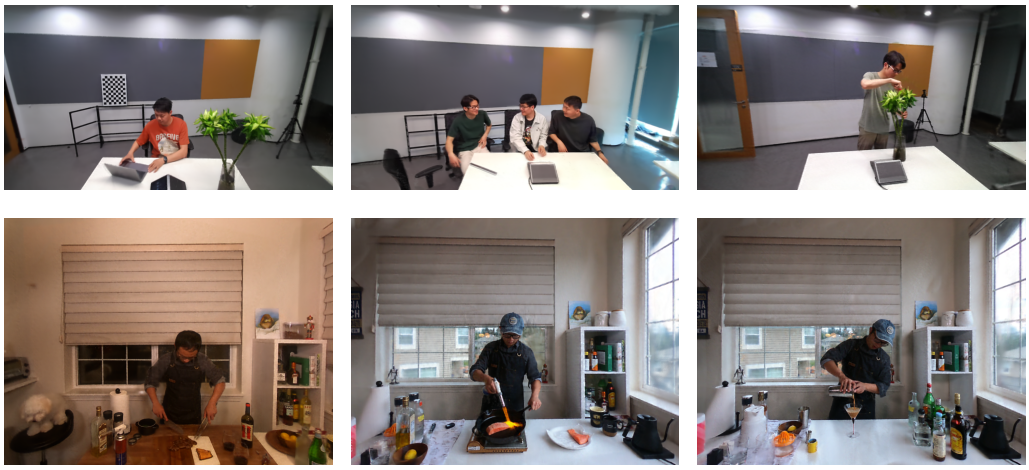


Figure 1: Rendering results on test view of **Top**: Our meet room dataset and **Bottom**: N3DV dataset. These novel view result are rendered at interactive speed ($\sim$10 FPS) by StreamRF.

framework, enabling it to tackle on-the-fly video sequences in a streaming manner. Inspired by the recent success achieved by Plenoxel [9] on training and inference efficiency, our method is built on learning of explicit grid representations. Specifically, the overall training objective is decomposed to complete training of a basic voxel grid model at the first frame with an incrementally tuning on model difference in subsequent frames. Only the model differences are expected to be trained and stored for each frame. A complete grid model given a time step i is simply attained by adding the model difference to the previous model at $i-1$ (or accumulating all the model differences before i with the basic grid model). By exploiting the prior of exhibiting local continuity between adjacent frames, we present a narrow-band tuning strategy for capturing and training the model difference efficiently, associated with a compression strategy for significantly reducing memory cost which may plague most explicit-grid methods. To further improve the training efficiency, we introduce a curriculum learning paradigm for learning pilot models to guide the optimization of full-scale models. The overall framework is simple yet efficient. Quantitative and qualitative experimental results validate the effectiveness of our approach. The approach can achieve the training speed of 10 seconds and inference time of 100 ms per frame for rendering a 1k resolution image with competitive quality, achieving 1,500 times speedup over the state-of-the-art dynamic neural radiance field methods (costing 56 GPU-days for training a sequence with 300 frames). It is also able to obtain 135 times acceleration with a much lower storage cost compared to per-frame explicit-grid benchmarks. The empirical analysis also demonstrates that using such a proper incremental training mechanism would not degrade the performance in the scenarios of dealing with relatively long sequences.

2 Problem Formulation

Neural Radiance fields (NeRF) [5] model a *static* scene implicitly by learning a continuous function mapping from a 3D spatial position $\mathbf{x} \in \mathbb{R}^3$ specified in a viewing direction $\mathbf{d} \in \mathbb{R}^3$ (represented as Cartesian unit vectors) to RGB radiance color $\mathbf{c} \in \mathbb{R}^3$ and opacity $\sigma \in \mathbb{R}$. The mapping function is typically implemented by multilayer perceptron (MLP) networks, $f_m : (\mathbf{x}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma)$, trained with a set of images with known camera pose. Given a ray $\mathbf{r}$ casting on the image plane through a pixel, by using a classical volume rendering technique [10], the expected color $\hat{C}(\mathbf{r})$ of the pixel is rendered by integrating the opacity and the color of sampling points along the ray:

$$\widehat{C}(\mathbf{r}) = \sum_{i=0}^{N-1} T_i \left(1 - \exp(-\sigma_i \delta_i)\right) \mathbf{c}_i, \quad T_i = \exp\left(-\sum_{j=0}^{i-1} \sigma_j \delta_j\right), \tag{1}$$

where δ_i denotes the distance to adjacent points. These methods are subjected to slow training and rendering speed, since ray casting involved in both training and rendering stage requires hundreds

of forward propagation (and backward propagation in training) through the MLPs for the sampling points on each ray. For example, JaxNeRF[11], though optimized through parallelization, needs over 20 seconds to render with at least one day of training.

When extending the mechanism to model a *dynamic* scene, one straightforward solution is to embed the time variable t into the representation, i.e., $f_t : (\mathbf{x}, \mathbf{d}, t) \mapsto (\mathbf{c}, \sigma)$, where the training time would linearly increase with respect to t. Training a video with NeRF frameworks may cost over 2,000 GPU-days. Although some variants, e.g., N3DV [8], improve training efficiency by virtue of sophistical designs, it still needs about 56 GPU-days in training to obtain a model with reliable rendering quality. More importantly, these implicit formulations are naturally suitable for offline processing which feeds all the (key)frames in advance for training, and may be not easily generalized to online training scenarios.

On the other hand, recent progress, representative by Plenoxel [9], has demonstrated compelling improvement on training and render efficiency for modelling a *static* scene. Specially, a sparse voxel grid is leveraged to replace the implicit representations in NeRF so that the scene is represented explicitly. Each voxel in the grid consists of the scalar opacity σ and the coefficient vector $\mathbf{k}$ of spherical harmonics (SH) for describing color information, which are directly optimized without neural networks. The color at the voxel v is obtained by,

$$\mathbf{v}_c = c(\mathbf{d}, \mathbf{k}) = \xi \left(\sum_{l=0}^{l_{max}} \sum_{m=-l}^{l} \mathbf{k}_l^m \cdot Y_l^m \cdot \mathbf{d} \right), \tag{2}$$

where ξ is the sigmoid function and $\{Y_l^m\}$ denotes the set of SH basis. To render the expected color of the pixel through a ray, the opacity and the color of a sampling point $\mathbf{x}$ is calculated by performing trilinear interpolation among the voxels within its neighborhood $\mathcal{N}(\mathbf{x})$,

$$\sigma = U_{\mathbf{v} \in \mathcal{N}(\mathbf{x})} \left(\mathbf{v}_\sigma \right), \quad \mathbf{c} = U_{\mathbf{v} \in \mathcal{N}(\mathbf{x})} \left(\mathbf{v}_c \right). \tag{3}$$

Then the pixel color is produced by integrating the sampling points along the ray based on Eq. 1. The voxels with zero opacy (or truncated by a minimal threshold) are pruned after sparsification, and an occupancy mask $\mathbf{M}$ with binary values is used for indicating if the space is occupied. More details can refer to the paper of Plenoxels [9]. In theory, the mechanism could significantly reduce the overall computational complexity as it omits the requirement of numerous MLP-based costly computation during training and inference. It is able to train a static scene with good rendering quality within 15 minutes on single GPU and render a $1k$ resolution image in 100 ms. These efficiency gains pose the chance to develop a highly efficient framework for modeling and rendering a dynamic scene.

3 Streaming Radiance Fields for Dynamic Scenes

To realize sparse grids for the modeling of a dynamic scene, one intuitive way is to directly extend 3D spatial dimensions (denoted by $H \times W \times D$) with the 4-th dimension indicating time t for frames. However, using explicit grids for a static scene may have faced the challenge on storage, although progressive training and sparsifying (i.e, pruning off empty voxels) [9] are used to alleviate the issue of costly storage. Simply expanding the grid into a space-time manner would make the storage overhead increase linearly with t, which is infeasible for handling videos with long sequences. On the other hand, per-frame training individually without considering temporal correlation may be feasible for achieving a tradeoff between storage and efficiency, while it is still far from the objective of pursing a high-performing 3D video synthesis framework.

To this end, we propose a novel method, named StreamRF, that realizes a streaming radiance field for dynamic scene rendering. Our approach is built on the benefits of utilizing an explicit representation via sparse voxel grids. Particularly, by leveraging some intrinsic priors exhibiting in videos, we develop an incremental training framework which enables a faster training convergence and lower storage over per-frame grid modelling.

Videos of forward-facing scenarios are typically composed of time-invariant components (e.g., static background) and time-variant components (e.g., moving objects) with small variation exhibiting

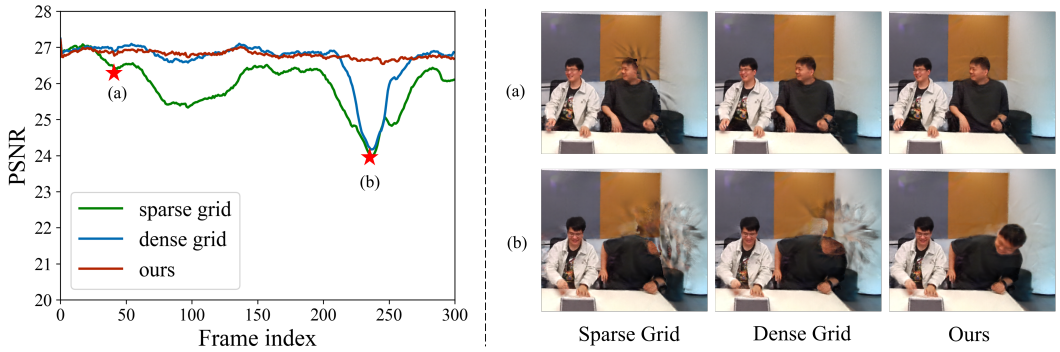


Figure 2: **Left:** PSNR comparison between tuning with (1) original sparse grid (2) dense grid and (3) our narrow band finetune in a 300 frames sequence. **Right:** visual comparison of above methods on (a) $40th$ frame (b) $240th$ frame. All results are trained with the same initial model.

local continuity between adjacent frames. By exploiting the fact, we decompose the sequential modeling problem into two sub-problems. We first fully train a base grid model given the first frame, denoted as $\mathbf{V}^0 = \{\mathbf{V}_c^0, \mathbf{V}_\sigma^0\}$ which stores the corresponding color features (the coefficients of spherical harmonics) and opacity values. Then the model is adapted to succeeding frames through the use of an effective tuning strategy. Per-frame tuning is expected to perform only on the changes between adjacent frames. The grid model in the current frame is produced by adding the model difference, which is optimized and stored during training phase, to the grid model in the previous frame. Formally, assume the model difference between the $i-1$-th and i-th frames is denoted as $\delta_i = \Delta(\mathbf{V}^i, \mathbf{V}^{i-1})$. In the inference phase, the grid model at the time step i can be obtained by,

$$\mathbf{V}^i = \mathbf{V}^{i-1} + \delta_i = \mathbf{V}^0 + \sum_{j=1}^{i} \delta_j. \tag{4}$$

Such a incremental formulation indeed presents an online training paradigm, enabling the proposed framework to handle on-the-fly video sequences.

3.1 Tuning with Narrow Bands

A problem arises: how to effectively capture the model difference between adjacent frames $i-1$ and i. Given the sparse grid model $\mathbf{V}_{i-1}$, any object motion at the time step i may incur changes on the opacity or colors of a part of voxels. It is hence likely to happen that informative points occur in the originally empty space as well as some voxels that are originally occupied becomes empty. Directly tuning the sparse grid can only change or one-way prune the non-empty voxels, intractable to handling long sequences.

To address the issue, we propose a tuning strategy based on narrow bands. We first retrieve and activate the narrow band for frame i, i.e., the region of interest in the grid $\mathbf{R}^i$, based on the occupancy mask $\mathbf{M}^{i-1}$ with binary values which indicates non-empty voxels at the frame $i-1$,

$$\mathbf{R}^i = \mathcal{F}_{di}(\mathbf{M}^{i-1}, \rho_d) \oplus \mathcal{F}_{er}(\mathbf{M}^{i-1}, \rho_e), \tag{5}$$

where $\mathcal{F}_{di}$ and $\mathcal{F}_{er}$ correspond to dilation and erosion operations, and ρ_d and ρ_e denote the corresponding radius parameters. $\oplus$ denotes the element-wise xor operation on the two masks to extract the additional region of the dilated area compared to the eroded one.

The narrow band actually represents a thin area around the surface of scenes (or objects in a composite scene). We use it for two purposes:

- For every empty voxels fall into the band, we restore them to the grid and initialize the opacity and color features with zero values.

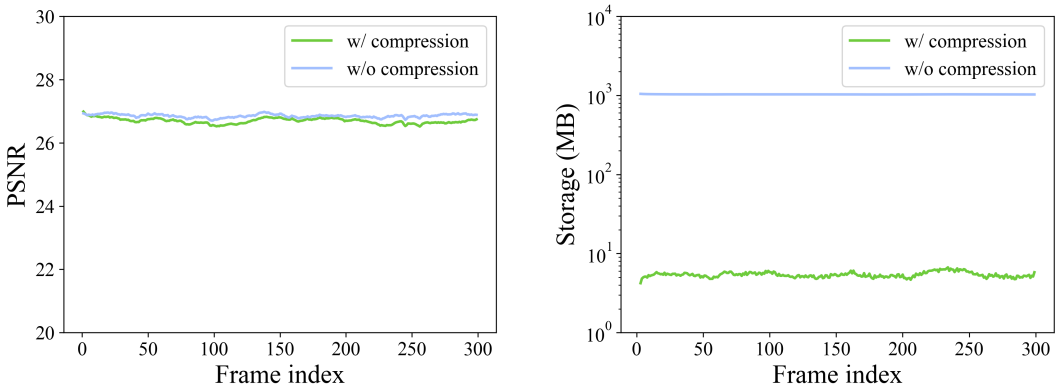


Figure 3: **Ablation study of diff-based compression**. We compare per frame PSNR (**left**) and storage size (**right**) with diff-based compression enabled and disabled. The storage space drops to 0.5% of the original with a negligible difference on PSNR (decrease 0.156 in average)

- Only the voxels within the band are tuned and the rest ones (with a large number) are not involved during the training stage for the frame i.

We also prune empty voxels to get a compact grid $\mathbf{V}_i$ with an occupancy mask $\mathbf{M}_i$ after tuning in the frame i.

The key idea of the narrow band-based tuning is built upon leveraging the intrinsic priors that motions induced by moving objects typically show small changes and local continuity within neighboring areas between adjacent frames. These assumptions have been successfully exploited in a wide range of video processing [12] and understanding tasks [13, 14], e.g, traditional mean-shift method for tracking [15]. We demonstrate that the strategy can effectively benefit to training convergence, capable of achieving the speed of per-frame $\sim$ 10s training for achieving a competitive quality (shown in section 5.4). Compared to the state-of-the-art implicit method [8] with the training speed of 4.5 GPU-hours for each frame in average (costing about 56 GPU-days for training a sequence of 300 frames), our method can achieve over 1,500 times speedup on training and over 200 times speedup on rendering efficiency (25 s VS 121 ms).

3.2 Difference-based Compression

Using explicit grid modelling yet poses the issue of large storage cost. Taking a grid size ($1408 \times 1156 \times 128$ in the experimental setting) for example, the overall storage costs about 5.4 MB after sparsity for one-frame model, as a conventional sparse grid. By virtue of narrow band tuning, we only need to store the model difference δ_i with a significantly smaller size over the complete grid model. The change of storage cost only happen in the following two scenarios, i.e., when adding voxels into the grid (i.e., appearing in the empty space) and pruning voxels from the grid (i.e., the previously occupied space become empty). In this regard, we introduce a difference-based compression strategy to further reduce the storage overhead. Particularly, we can efficiently find out the change areas by using the occupancy masks $\mathbf{M}^{i-1}$ and $\mathbf{M}^i$ between the adjacent frames $i-1$ and i,

$$\begin{aligned} \mathbf{M}_a^i &= \mathbf{M}^i \odot (1 - \mathbf{M}^{i-1}), \\ \mathbf{M}_e^i &= \mathbf{M}^{i-1} \odot (1 - \mathbf{M}^i), \\ \mathbf{M}_r^i &= \mathbf{M}^{i-1} \odot \mathbf{M}^i \odot \ \mathbf{1}\{\gamma(\delta_i^c) > \xi\}, \end{aligned} \tag{6}$$

where $\mathbf{M}_a^i$, $\mathbf{M}_e^i$ and $\mathbf{M}_r^i$ representatively denote the masks of adding voxels, erasing voxels and remaining ones while the features at the voxels may change. $\gamma(\cdot)$ defines the function of getting the L1 norm along the axis of color features (SH coefficients) on the model difference, and ξ denotes a threshold. $\mathbf{M}_r^i$ actually involves the voxels with relatively large modification. Instead of storing model difference δ_i directly, we store the three masks, the voxel representations (SH coefficients and

opacity) belonging to the adding area $\mathbf{M}_a^i$ and the representation differences for the voxels within the remaining area $\mathbf{M}_r^i$. It can achieve a remarkable reduction rate (187 times) on storage that reduces **1015 MB** to **5.4 MB** for each frame in average.

4 Efficient Training with Pilot Model Guidance

To further improve the training efficiency of StreamRF, we exploit a curriculum learning paradigm that trains from an easier problem. Formally, given the grid model $\mathbf{V}^{i-1}$, we adopt down-sampling operations to get a *pilot* model of a smaller size with the scale S,

$$\widehat{\mathbf{V}}^{i-1} = \downarrow^S (\mathbf{V}^{i-1}). \tag{7}$$

When setting the S to $1/2$, the capacity of pilot model is roughly $1/8$ of the full grid. Training on a smaller model would reduce the computational complexity at every iteration (with the same batch size) and also ease optimization. The model difference on the pilot models $\widehat{\delta}_i = \Delta(\widehat{\mathbf{V}}^i, \widehat{\mathbf{V}}^{i-1})$ are implemented by using $\{\widehat{\mathbf{M}}_a^i, \widehat{\mathbf{M}}_e^i, \widehat{\mathbf{M}}_r^i\}$ as following Eq. 6. Then we can get a guidance mask $\widehat{\mathbf{G}}^i$ by merging the three mask via elementwise "or" operator and upsampling to the full scale,

$$\begin{aligned} \widehat{\mathbf{G}}^i &= \widehat{\mathbf{M}}_a^i \vee \widehat{\mathbf{M}}_e^i \vee \widehat{\mathbf{M}}_r^i, \\ \mathbf{G}^i &= \uparrow^S (\widehat{\mathbf{G}}^i). \end{aligned} \tag{8}$$

We adopt the well-trained pilot model with the guidance mask for fine-tuning the full-scale grid model. All the voxels outside the guidance mask $\mathbf{G}^i$ are frozen without optimization when training at the frame i.

We observe that using this strategy also facilitate pursuing a higher fidelity model compared to directly training on the full-scale grid with the same training times. Specially, the qualitative results (in Fig. 4) reveals that the model via directly training on the full-scale grid may encounter flicker issue when rendering along time steps to a sequence, i.e., sparkling noise may occur in the stable background area, when training a short period. The issue could be addressed by lengthening the training schedule with a small learning rate. While training with the pilot model guidance can effectively prohibit the flicker artifacts from happening. Training first on a down-sampled pilot grid essentially introduces local consistency for neighboring voxels when upsampling to the full-scale grid, enabling a more stable optimization for the streaming paradigm.

5 Experiments

5.1 Datasets

Meet room Dataset. We build a multi-view capture system using 13 Azure Kinect cameras. Twelve cameras were used for training, and one camera was used for testing. We use an external pulse signal to synchronize all camera shutter. Though this system is capable of depth capture, we do not use depth results in this research. The capture videos were recorded at a resolution of 1280×720 and a frame rate of 30 FPS. The total size of the capture system was 100 cm $\times$ 75 cm, making it much sparser than some common datasets such as LLFF and N3DV. We will release this dataset for research purposes.

Neural3D Video Dataset[8] contains 6 dynamic indoor scenes with varying illuminations, view-dependent effects, and highly volumetric effects. Videos were captured at a resolution of 2704×2078 and a frame rate of 30 FPS. Following the N3DV paper, we use one view for testing and 18 views for training. We choose the designated sequence and downsample it to 1352×1039 for a fair comparison of the quantitative score.

5.2 Implementation Details

Initialization. We follow the standard training protocol in a forward-facing scene of Plenoxels[9]. We start by $256 \times 256 \times 128$ grid, upsample and sparsify for every 38.4K iterations until the model

Methods	N3DV			Meet Room		
	PSNR↑	Train time↓	Inference time↓	PSNR↑	Train time↓	Inference time↓
JaxNeRF*[11]	28.53	485 mins	67 s	27.11	473 mins	40 s
LLFF*[18]	23.23	8 mins	**0.004 s**	22.88	3 mins	**0.003 s**
N3DV[8]	**29.58**	260 mins	>67 s	-	-	-
Plenoxels*[9]	28.68	23 mins	0.12 s	**27.15**	14 mins	0.1 s
Ours	28.16	**0.17 mins**	0.12 s	26.79	**0.17 mins**	0.1 s

* Denote train from scratch per frame. The gray numbers are directly taken from the corresponding paper or estimated.

Table 1: **Comparison with related work on N3DV and Meet Room dataset.** we report per frame training time for better comparison, all results except N3DV are recorded with our 3090 GPU. Our method achieves a comparable PSNR with a significantly faster per frame optimization time.

reaches the final resolution of $1408 \times 1156 \times 128$. The total training takes 128K iterations with a batch size of 5,000 rays. We adopt the RMSProp [16] optimizer with a decay parameter of $\beta = 0.95$ for voxel grid optimization. We also adopt a higher TV loss to mitigate the foggy issue, with λ_{TV} set to 5×10^{-4} for opacity and 5×10^{-3} for color features.

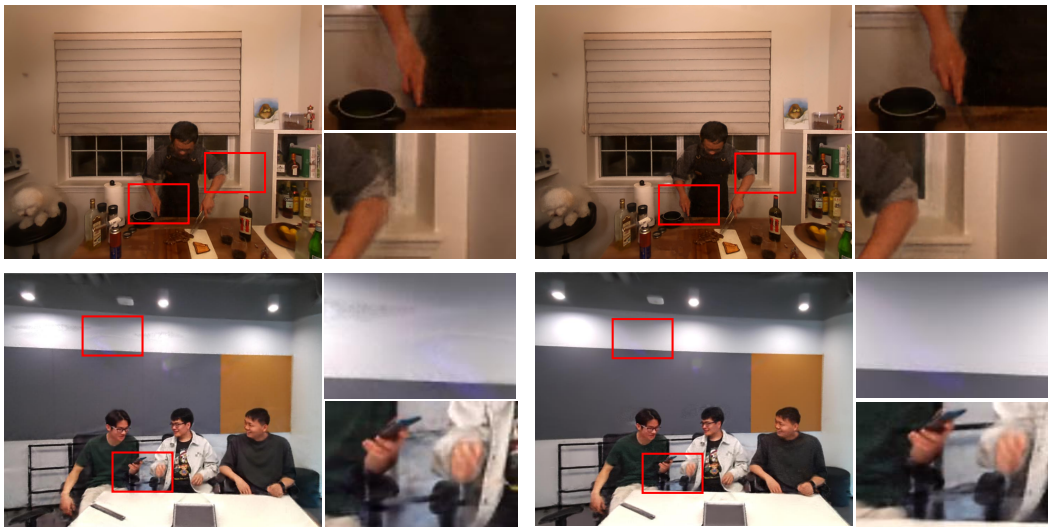
Per Frame Tuning. The main model and the pilot model are trained in parallel. We train the main model with 500 iterations and the pilot model with 1,000 iterations. We use RMSprop to optimize both models with a batch size of 20,000 rays. To promote the pilot model to have a better convergence, we enlarge λ_{TV} 10 times when tuning the pilot model while keeping it untouched for the main model.

Diff Based Compression. We set the SH coefficient threshold ξ to 1/27 (27 is the channel number of SH coefficients). A smaller choice would drastically increase saving size with only marginal gain in PSNR. We convert all saving SH coefficient and density value to float16 to further decrease saving size. In actual implementation we could merge $\mathbf{M}_a^i$ and $\mathbf{M}_e^i$ to one single mask to save more space. Moreover, as the mask is mostly consecutive 0 and 1, both masks have a very high compression ratio (over 99%) even with standard compression library (*e.g.* ZLIB[17]). We pack all tensors together and compress them with ZLIB to get the final on-disk per frame storage.

5.3 Results

We first quantitatively compare with benchmark methods in Table 1. We use peak signal-to-noise ratio (PSNR), per frame training time, and inference time as evaluation metrics. We follow the N3DV testing protocol [8] and choose "flame_salmon" as the test sequence. We evaluate the four methods every ten frames to calculate the mean value of per-frame metrics. We refer to the values of PSNR and training time reported in the paper of N3DV for a fair comparison. We estimate the rendering time of N3DV according to its network design. Because they use a larger MLP than the one used in NeRF [5], its inference time is supposed to be longer than NeRF.

As shown in the table, the two implicit methods JaxNeRF and N3DV reveal high computation cost that requires over 8 GPU hours and 4 GPU hours for per-frame training (N3DV costs about 56 GPU days to train 300 frames), respectively. As for testing speed, the LLFF achieve the fastest record. Because it only needs blending the multi-plane image to render a novel view. However, it fails to produce a satisfactory result. We achieve the optimization speed of costing 10 seconds only, outperforming all the baseline methods in terms of training efficiency. Meanwhile our method can achieve a comparable PSNR to the previous SOTA. For the meet room dataset, we use the same sequence as in Fig 2 for testing and following the same test protocol as above. Because our dataset contains fast motion and camera spacing is larger than N3DV dataset, we could see a clear PSNR drop on all four methods. Nevertheless, our method still achieves a comparable performance with per frame training counterpart, around 0.3 drop in PSNR compared to JaxNeRF and Plenoxels, while obtaining the acceleration over $2,700\times$ and $80\times$, respectively. More experimental results and analysis can be referred to supplementary materials.



w/o Pilot Model Guidance w/ Pilot Model Guidance

Figure 4: **Ablation study of pilot model guidance:** we compare **Left:** StreamRF trained Without pilot model guidance and **Right:** with pilot model guidance in both N3DV dataset (top) and meet room dataset (bottom). These results reflect that pilot model can help reduce artifacts in both static background and dynamic foreground.

5.4 Ablation study

Narrow Band Finetune We compare the narrow band with the following two counterparts, both quantitative and visually: (1) directly adopt the original sparse voxels grid, (2) switch the original sparse voxels grid to its dense counterpart. It can be seen in Fig.2. The sparse grid failed immediately (Fig.2 (a)), though the right person only move at a small distance. This artifact is because the subject has moved out of the trainable area. A dense voxel grid allows every voxel to be trainable. It still fails when the right person has a large movement(Fig.2 (b)). This artifact could be caused by the rapidly increasing number of active voxels after switching from sparse to dense (5% vs. 100% of all voxels). Both PSNR curve and rendering result show our method achieves comparable PSNR with a dense grid and can handle large movement in a dynamic scene. Moreover, in this ablation study, our method trains around 40% faster and requires 63% less memory consumption than dense finetune.

Diff-based Compression We validate the performance and effectiveness of diff-based compression in Fig. 3. In this ablation, we first train our StreamRF on an entire sequence. The trained StreamRF is then saved with diff-based compression enabled and disabled. We then compare the PSNR and saving size between the two test view sequences. We present the per frame PSNR to show whether diff-based compression impairs the performance in the long run.

The quantitative result shows that our compression method has negligible impact after a very long propagation. Besides deducting overall storage size from 1015 GB to an average of 5.4 MB for each frame, we can observe that performance of our method does not downgrade with time and the incremental storage for each frame is roughly stable along time steps.

Pilot Model Guidance We validate our pilot model guidance visually in Fig. 4. For base setting (right), we train the pilot model with 1,000 iterations and the full model with 500 iterations. For the ablation setting (left), we directly train the full model with 1,000 iterations to make per frame training time equivalent in both settings. As shown in Fig. 4, pilot model guidance help reduce various kinds of artifacts: (top) vanishing hands and blurry elbow. (bottom) sparkle noise in a white wall, the fuzzy area between two people. These results verified our intention that we could use an easy-to-optimized small model to provide a more optimal initialization to the full model.

6 Related Work

Static Novel View Synthesis Novel view synthesis from a set of input images with known camera poses has been studied intensively in the field of computer vision and computer graphics. Earlier works like Lumigraph [19, 20] and Light-Fied [21, 22] focus on interpolating the sampled light rays for realistic rendering. It has been proved [23, 24] that the sampling density heavily depends on the underlying geometry of the captured scene. Thus, many works strive to recover an approximate [25, 26] or precise geometry model [27] to facilitate novel view rendering. Among those works, multi-plane image (MPI) [18, 28–30] is a promising approximate geometry model that enables realistic rendering of generic scenes.

More recently, NeRF [5] represents the radiance field with a multi-layer perceptron (MLP) network and combines it with differentiable volume rendering for view interpolation. This approach achieved tremendous success and there are many variants to push its limits towards every aspect. The works [6, 31–33] focus on improving rendering quality, while the works [34–36] strive to accelerate rendering speed, and other works [9, 37–39] propose to replace implicit function by an explicit or hybird counterpart to boost training convergence.

Dynamic Novel View Synthesis Novel view synthesis has also been heavily studied with dynamic scenes. Early effort of generating 3D video can trace back to the 90s [1, 40], which computes explicit depth maps to facilitate novel view interpolation. This approach has been combined with more sophisticated capturing hardware to generate streamable free-viewpoint video in [2]. Matusik and Pfister [41] further integrate camera arrays and 3D displays to build an end-to-end 3D TV system.

More recent works learn a neural network to capture the radiance field of a dynamic scene. Some of them [42–44] focus on monocular video input for novel view synthesis at nearby viewpoints. Li et al. [42] generalize the original NeRF model to a neural scene flow field to deal with dynamic scenes. Xian et al. [43] bring depth prior to better estimate the 4D radiance filed. Gao et al. [44] jointly train an dynamic NeRF and static NeRF to better preserve details at both part. In compraison, Broxton et al. [3] and Attal et al. [45] use multi-camera stereo rig and extend MPI (Multi Plane Image) to MSI(Multi Sphere Image) to support view interpolation with larger field of view. All these above methods only support small movement of camera viewpoints.

Bansal et al. [46] use multiple mobile phones to capture a dynamic scene and compose the static and dynamic components with neural networks for 4D space-time visualization. The closest work to our setting is Neural 3D Video (N3DV)[8], which directly extends NeRF with an additional timecode input, and accelerates computation with a hybrid importance sampling and hierarchical training. N3DV can produce strong visual results given enough optimization. However, N3DV is very time consuming, requiring over 5 GPU hours per frame in training, and 10+ seconds per frame in rendering, making it impractical for real-world applications.

7 Conclusion

We propose a novel streaming radiance field method for effectively reconstructing and rendering dynamic scenes with explicit grid representations. The modelling of dynamic scenes is formalized as an incremental learning paradigm which allows the method for handling on-the-fly video sequences without need of recording scenes in advance. By virtue of narrow band tuning, our approach can achieve a fast training convergence. The storage cost induced by using explicit grids can be obviously reduced by performing difference-base compression. We also present an efficient training mechanism with pilot model guidance to further improve model optimization. Experiments demonstrate that our approach is capable of training a high-performing model for dynamic scenes with the speed of 10s for tuning every frame, achieving over $1500\times$ speedup over the state-of-the-art implicit dynamic methods. As a future direction, we expect to pursue further acceleration to support real-time training for streaming dynamic scenarios.

References

[1] C. Lawrence Zitnick, Sing Bing Kang, Matthew T. Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *international conference on computer graphics and interactive techniques*, 2004.

[2] Alvaro Collet, Ming Chuang, Patrick Sweeney, Don Gillett, Dennis Evseev, David J. Calabrese, Hugues Hoppe, Adam G. Kirk, and Steve Sullivan. High-quality streamable free-viewpoint video. *international conference on computer graphics and interactive techniques*, 2015.

[3] Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew DuVall, Jason Dourgarian, Jay Busch, Matt Whalen, and Paul Debevec. Immersive light field video with a layered mesh representation. 39(4):86:1–86:15, 2020.

[4] Yizhong Zhang, Jiaolong Yang, Zhen Liu, Ruicheng Wang, Guojun Chen, Xin Tong, and Baining Guo. Virtualcube: An immersive 3d video communication system. *CoRR*, abs/2112.06730, 2021.

[5] 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.

[6] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arxiv CS.CV 2010.07492*, 2020.

[7] Ky Giang Nguyen and Dietmar Saupe. Rapid high quality compression of volume data for visualization. In *Computer Graphics Forum*, volume 20, pages 49–57. Wiley Online Library, 2001.

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

[9] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021.

[10] James T. Kajiya and Brian Von Herzen. Ray tracing volume densities. *international conference on computer graphics and interactive techniques*, 1984.

[11] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an efficient JAX implementation of NeRF, 2020.

[12] Emanuele Trucco and Konstantinos Plakas. Video tracking: a concise survey. *IEEE Journal of oceanic engineering*, 31(2):520–529, 2006.

[13] Miki Haseyama, Takahiro Ogawa, and Nobuyuki Yagi. [survey paper] a review of video retrieval based on image and video semantic understanding. *ITE Transactions on Media Technology and Applications*, 1(1):2–9, 2013.

[14] Gal Lavee, Ehud Rivlin, and Michael Rudzsky. Understanding video events: A survey of methods for automatic interpretation of semantic occurrences in video. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(5):489–504, 2009.

[15] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 2, pages 142–149. IEEE, 2000.

[16] Geoffrey Hinton. RMSProp.

[17] Jean loup Gailly and Mark Adler. zlib, 2017.

[18] 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*, 2019.

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

[20] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael Cohen. Unstructured lumigraph rendering. *international conference on computer graphics and interactive techniques*, 2001.

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

[22] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. In *ACM SIGGRAPH*, 1999.

[23] Jinxiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. In *ACM SIGGRAPH*, 2000.

[24] Heung-Yeung Shum and Sing Bing Kang. Review of image-based rendering techniques. *visual communications and image processing*, 2000.

[25] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3d photography. In *ACM SIGGRAPH*, 2000.

[26] Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. Casual 3d photography. *ACM Transactions on Graphics*, 2017.

[27] Paul Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *ACM SIGGRAPH*, 1996.

[28] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. *Computer Vision and Pattern Recognition*, 2020.

[29] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. *Computer Vision and Pattern Recognition*, 2019.

[30] Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew DuVall, Jason Dourgarian, Jay Busch, Matt Whalen, and Paul Debevec. Immersive light field video with a layered mesh representation. *ACM Transactions on Graphics*, 2020.

[31] 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. *arXiv: Computer Vision and Pattern Recognition*, 2021.

[32] 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. *CVPR*, 2022.

[33] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[34] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021.

[35] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien P. C. Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *ICCV*, 2021.

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

[37] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

[38] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. 2022.

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

[40] Takeo Kanade, Peter Rander, and P. J. Narayanan. Virtualized reality: constructing virtual worlds from real scenes. *IEEE MultiMedia*, 1997.

[41] Wojciech Matusik and Hanspeter Pfister. 3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *international conference on computer graphics and interactive techniques*, 2004.

[42] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. *arXiv: Computer Vision and Pattern Recognition*, 2020.

[43] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. *Computer Vision and Pattern Recognition*, 2021.

[44] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. *arXiv: Computer Vision and Pattern Recognition*, 2021.

[45] Benjamin Attal, Selena Ling, Aaron Gokaslan, Christian Richardt, and James Tompkin. Matryodshka: Real-time 6dof video view synthesis using multi-sphere images. In *European Conference on Computer Vision*, pages 441–459. Springer, 2020.

[46] Aayush Bansal, Minh Vo, Yaser Sheikh, Deva Ramanan, and Srinivasa G. Narasimhan. 4d visualization of dynamic events from unconstrained multi-view videos. *Computer Vision and Pattern Recognition*, 2020.