© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes,creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. https://dx.doi.org/10.1109/CVPR52729.2023.01195

NeRFLight: Fast and Light Neural Radiance Fields using a Shared Feature Grid

Fernando Rivas-Manzaneque[1,2] Jorge Sierra-Acosta[1] Adrian Penate-Sanchez[3]
Francesc Moreno-Noguer[4] Angela Ribeiro[5]

[1]Arquimea Research Center, [2]Universidad Politécnica de Madrid
[3]Universidad de Las Palmas de Gran Canaria, [4] Institut de Robòtica i Informàtica Industrial, CSIC-UPC
[5] Centre for Automation and Robotics, CSIC-UPM

Abstract

While original Neural Radiance Fields (NeRF) have shown impressive results in modeling the appearance of a scene with compact MLP architectures, they are not able to achieve real-time rendering. This has been recently addressed by either baking the outputs of NeRF into a data structure or arranging trainable parameters in an explicit feature grid. These strategies, however, significantly increase the memory footprint of the model which prevents their deployment on bandwidth-constrained applications. In this paper, we extend the grid-based approach to achieve real-time view synthesis at more than 150 FPS using a lightweight model. Our main contribution is a novel architecture in which the density field of NeRF-based representations is split into N regions and the density is modeled using N different decoders which reuse the same feature grid. This results in a smaller grid where each feature is located in more than one spatial position, forcing them to learn a compact representation that is valid for different parts of the scene. We further reduce the size of the final model by disposing of the features symmetrically on each region, which favors feature pruning after training while also allowing smooth gradient transitions between neighboring voxels. An exhaustive evaluation demonstrates that our method achieves real-time performance and quality metrics on a pair with state-of-the-art with an improvement of more than 2$\times$ in the FPS/MB ratio.

1. Introduction

The use of 3D objects reconstructed from real images is becoming popular in a number of applications, such as virtual-reality or online video-games. The increasing need for realistic elements makes image-based reconstruction an adequate alternative to modeling these objects from scratch using fully manual or semi-automatic design engines. At the same time, however, the larger the number of these assets is, the higher the constrain of their size in order to meet certain bandwidth and storage requirements.

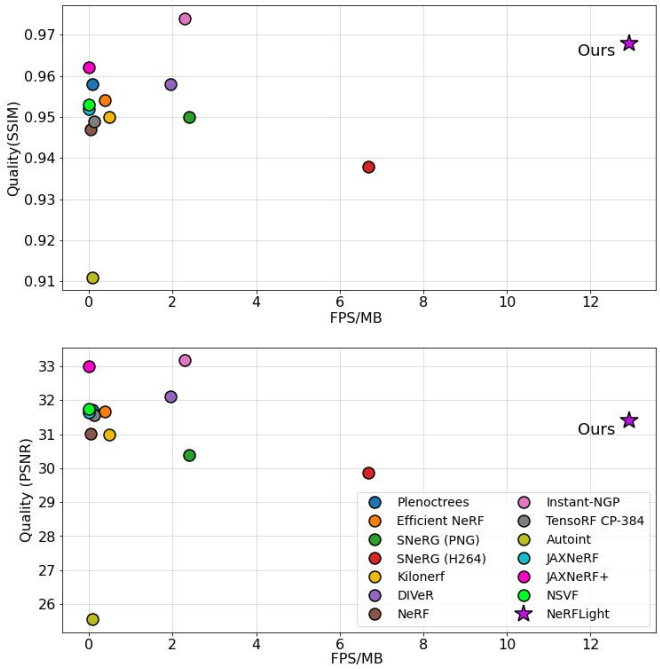


Figure 1. NeRFLight is able to double the FPS/MB ratio of the second best method while obtaining similar quality metrics to state of the art.

Methods based on neural rendering have arisen as a promising approach to solve this challenge. In particular, Neural Radiance Fields (NeRF) [25] have demonstrated outstanding potential. Instead of modeling the plenoptic function [1] by means of explicit geometry representations such as point clouds [2] or voxels [22], NeRF uses a coordinate-based multi-layer perceptron (MLP) to model a density and a color field that acts as a proxy of the plenoptic function. Applying this MLP in combination with positional encoding, NeRF achieves exceptional results in scene representation, improving the quality and compactness of

the previous methods [22, 24] and resulting in an explosion of variants [4,30,55]. However, one of the major limitations of NeRF-based methods is their low rendering speed, being a barrier to being used in applications that require real-time.

This limitation has been addressed in different works, by either "baking" the outputs of NeRF in a data structure [10,11,54] or using additional features located in an explicit grid [26, 48]. Nevertheless, all these real-time NeRF rendering methods sacrifice the compactness of the representation and require relatively large models, resulting in a poor render-time performance vs storage-cost ratio (FPS/MB). This prevents these models from being deployed in applications that, besides requiring real-time, have memory storage constraints.

In this work, we introduce NeRFLight, a new approach to reduce the size of grid-based NeRF models while achieving real-time view synthesis and retaining the rendering quality of slower and computationally expensive models. Our key idea consists in splitting the volume of the density field used in NeRF into eight different regions, each with a different decoder, but sharing a common feature grid. This allows an $8\times$ reduction in the number of total features. We also propose a novel configuration of the features that exploits the symmetry of neighboring voxel grids and favors a seamless reconstruction throughout all regions. This not only increases the rendering quality but also results in a more compact model. In addition, we also leverage the deterministic volume integration introduced in [48] to obtain a better representation than using Monte Carlo integration.

We perform an exhaustive evaluation on both synthetic and real data in terms of the FPS/MB ratio[1].

The NeRFLight architecture we propose achieves rendering speeds of up do 181 FPS with models of only 14MB. As shown in Fig. 1 this yields an FPS/MB metric that is $2\times$ larger than the closest approach in the state-of-the-art, while achieving a rendering quality (measured using PSNR and SSIM) on pair with much larger models.

To summarize, our main contributions are the following: **1)** We introduce a NeRF architecture based on several density fields and a shared feature grid that provides a light and fast representation; **2)** We devise an approach that enforces symmetry of neighboring voxels that favors a seamless reconstruction, increases the model accuracy and further reduces its size; **3)** All this results in the Neural Radiance Field implementation with the highest frame rate vs. storage cost ratio of the current state of the art.

2. Related work

Neural Volume Reconstruction: Novel view synthesis of 3D scenes is a well-known field in computer vision. In the last years, research has turned towards the use of neural network representations [2, 22, 24] in what has been called Neural Rendering or Neural Volume Reconstruction. Such neural networks are applied using intermediary explicit geometry representations like point clouds [2, 46], meshes [35, 36, 45], voxel grids using CNNs [22, 33, 38] or multi-plane images [24, 41, 57].

Recently, the use of neural fields has become popular in this task. Neural fields apply coordinate-based networks to encode the relation between spatiotemporal coordinates and a given physical magnitude (see [51] for a detailed review). To achieve novel view synthesis, different types of fields (magnitudes) have been used to act as proxies of the plenoptic function [1] of a given scene. For example, [13, 39, 53] apply signed distance fields and an appearance field (color). [27, 37] replace signed distance fields by occupancy fields. Neural Radiance Fields (NeRF) [25] proposed a simple yet accurate approach that achieved unprecedented results using density and color fields. Thus, most of the subsequent works have built upon this strategy, aiming to extend original NeRF to different situations, such as unbounded scene reconstruction [3, 34, 44, 50], scene relighting [4, 5, 40, 56], scene composition [16, 21, 52], dynamic scenes [8,17,18,28,30,47,49] or fast convergence [9,26,42].

Baking NeRF: Several approaches have addressed the problem of speeding up NeRF at inference. The first strategy consists in "baking" the output of a pre-trained NeRF. For instance, KiloNeRF [32] uses a pretrained NeRF to train a uniform spatial grid of small MLPs in a teacher-student manner. The original NeRF is then baked into a grid of tiny NeRFs that accelerate rendering, although the use of a large number of MLPs is very memory-demanding. [54] learns first a NeRF that predicts the factors of a spherical harmonic that are embedded into a PlenOctree structure capable of modeling the view-dependent effects, so no MLP evaluation is needed at inference. However, explicitly storing spherical harmonic coefficients for the whole scene is not memory efficient. Efficient-NeRF [11] achieves the fastest rendering speed by storing the density output of NeRF and generating an octree-based structure (NeRFTree). Despite the acceleration they provide, the large memory footprint of this structure and the density grid results in a low rendering frame rate vs storage ratio. SNeRG [10] directly tackles the reduction of memory footprint. For this purpose, this method bakes the alpha composition and color along the ray together with view-based features that are decoded by a small MLP at inference. The baked information is stored in a texture atlas, where image compression algorithms such as PNG or H264 are applied. However, this compression leads to relatively low-quality metrics.

Explicit feature grids: The last strategy, and possibly the most popular, consists in arranging a set of trainable pa-

[1] In a similar way as done in [10], where an evaluation metric of *FPS/Watts* was used to measure the compromise of *speed vs power consumption*.

rameters as an explicit feature grid, reducing the number of required operations at each model inference. These features are linearly interpolated to serve as an input to small and efficient MLPs. [48] leverages this interpolation to introduce a deterministic volume integration that leads to an accurate and fast-rendering model. Nevertheless, feature storage lowers the resulting FPS/MB metric. To reduce feature memory footprint, Instant-NGP [26] uses each feature at several spatial locations using a hash-grid and applies a multiresolution representation to deal with feature collision. It also applies half-precision for feature storage, improving the compactness of the model. However, the multiple linear interpolations needed due to the multiresolution hash-grid lead to a reduction in rendering speed. Finally, methods such as TensoRF [6], or VQAD [43] apply more elaborated feature transformations like tensor decomposition or vector quantization, respectively. These strategies produce very compact models, however, they also make feature retrieval more complex which prevents real-time rendering.

NeRFLight also relies on explicit feature grids. As in [48] we also apply deterministic integration to accelerate rendering and improve the accuracy of the reconstruction. On the other hand, we also apply features located at different regions of the space. However, instead of using a multiresolution scheme, we apply multiple density decoders (like [31,32]) to deal with feature collisions, avoiding the rendering speed loss of [26]. We do not make use of a hash grid. Instead, we have devised a more optimal symmetric configuration for the features that leads to high-quality seamless reconstruction and even faster and lighter models.

3. Background

NeRF [25] and its variants represent 3D scenes by approximating the plenoptic function as a density field $\sigma(\mathbf{x})$ and a color field $\mathbf{c}(\mathbf{x}, \mathbf{d})$, which vary depending on the spatial position $\mathbf{x}$ and viewing direction $\mathbf{d}$. These fields are encoded using a coordinate-based MLP with weights $\mathbf{w}$. Radiance is accumulated along a ray $\mathbf{r}(t) = \mathbf{o} + \mathbf{d}t$ using the volume rendering equation [12]:

$$\hat{\mathbf{c}}(\mathbf{r}) = \int_0^\infty e^{-\int_0^t \sigma(\mathbf{r}(\tau))d\tau} \sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \qquad (1)$$

to obtain the final color of a given pixel. Eq. (1) is approximated using Monte Carlo integration on a set of colors and densities coming from the inference of the MLP at n randomly sampled points along the ray ($\mathbf{c}_i, \sigma_i = \mathrm{MLP}_{\mathbf{w}}(\mathbf{x}_i, \mathbf{d}_i)$). Radiance and density are assumed to be constant between samples of a single ray. Thus, Eq. (1) can be approximated as:

$$\hat{\mathbf{c}}(\mathbf{r}) = \sum_{i=1}^{n} T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \mathbf{c}_i \qquad (2)$$

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)\,, \qquad (3)$$

where T_i corresponds to the accumulated transmitance at a given point and δ_i to the length of the interval between points $\delta_i = \|\mathbf{x}_{i+1} - \mathbf{x}_i\|_2$. NeRF-like models learn to approximate the density and color fields of the scene by minimizing the photometric loss between the pixels rendered using Eq. (2) and the pixels from the training images.

In order to accelerate rendering, some methods [20, 48] arrange additional training parameters as explicit features located at the vertices of a voxel grid that are used as input for the MLP. This allows to reduce its size and, since the number of parameters involved in each model inference is also reduced, a faster rendering speed is obtained. To achieve this, these features are linearly interpolated depending on the position $\mathbf{x}$ of the sampled point:

$$(\sigma_i, \mathbf{c}_i) = \mathrm{MLP}_{\mathbf{w}}\left(\hat{\mathbf{f}}(\mathbf{x}), \mathbf{d}\right), \qquad (4)$$

where $\hat{\mathbf{f}}(\cdot)$ is the trilinear interpolation of the features of a given voxel. Despite of the achieved acceleration, when using explicit feature grids, most of the compactness provided by NeRF's MLP is lost due to the storage required cost for the feature grid. This has a negative impact for the FPS/MB metric.

4. Method

As mentioned in Sec. 3, in order to reduce computational cost we represent a 3D scene as a density and a color field. Similar to [48], we apply deterministic volume integration and implicit feature initialization. We also use a coarse-to-fine strategy to speed up training and perform empty space pruning to accelerate model inference. However, unlike other grid-based methods, our representation is compact thanks to the proposed shared feature grid we describe below.

4.1. Shared Feature Grid

As shown in Fig. 2, the scene is split into N different regions. NeRFLight makes use of a shared feature grid that is repeated in each of them. Therefore, the features are arranged at several locations, forcing the information to be more compact as it needs to represent more than one part of the scene. At a given location, a region-specific density decoder θ_i is used to convert the corresponding features into the density value σ and an intermediate representation h. Finally, a single-color decoder uses this intermediate representation to obtain the color as a function of the viewing direction. Therefore, for a given point $\mathbf{x_j}$ in a ray $\mathbf{r_j}(t) = \mathbf{o_j} + \mathbf{d_j}t$ we have:

$$\sigma_j, h_j = \theta_i(\hat{\mathbf{f}}(\mathbf{x}_j)), \quad i \in [1, N]\,, \qquad (5a)$$

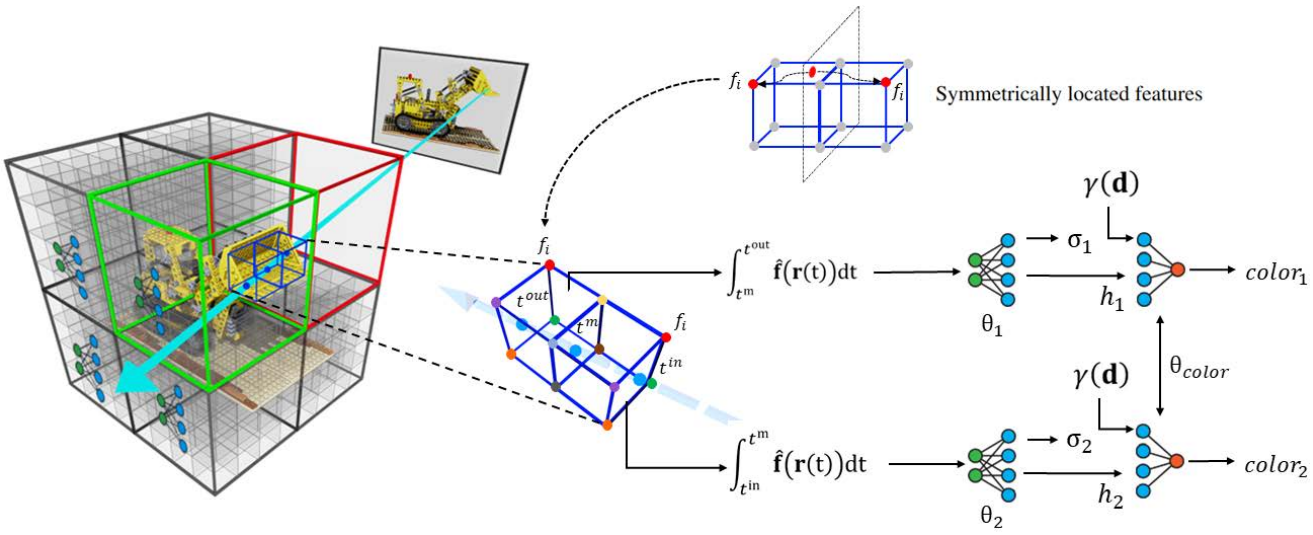


Figure 2. **NeRFLight overview.** Our method represents a scene as a set of N density fields and a single color field parameterized by N density decoders θ_i, a color decoder θ_{color} alongside an explicit feature grid. This feature grid is repeated for each of the regions, i.e, density fields, and is shared by all the decoders. The features are placed symmetrically to the scene center on each volume, which favours continuity between density fields and feature pruning after training.

$$\mathbf{c}_j = \theta_{color}(h_j, \gamma(\mathbf{d}_j)), \tag{5b}$$

where $\hat{\mathbf{f}}(\cdot)$ is the integrated feature we describe later in Sec. 4.3 and $\gamma(\mathbf{d_j})$ is the positional encoding of the viewing direction, as in [25]. Note that Eq. (5a) is equivalent to representing the scene using eight different density fields while, as depicted by Eq. (5b), a single color field is used. This representation allows for reducing the number of unique features in the voxel grid and, subsequently, the memory needed by the feature grid, by a factor of N. As a design choice, we consider each of the N regions to be a cube with a size equal to half the side of the bounding box, resulting in a total of $N = 8$ regions and their corresponding density decoders. For all the regions, the voxel grid resolution is kept constant and it is chosen independently for each dataset. The dimension of the features located at each vertex of these grids is set to 32.

4.2. Symmetric Voxel Grid

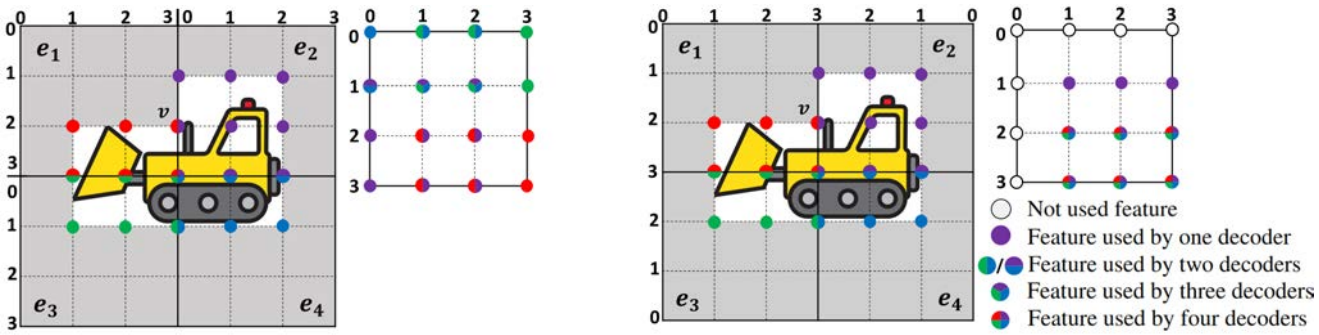
A natural approach when arranging the features in the voxel grid would be to simply repeat them in the same order for each region of the scene, using a linear configuration as shown Fig. 3a. However, this may introduce discontinuities at the seams between regions. Some previous works using neural fields solve this discontinuity issue by overlapping the different regions [23]. While this might be effective in a fully-implicit representation, doing this in an explicit grid of n^3 features would increase the number of features by $3n^2 + 3n + 1$, resulting in a growth of the model size and reducing the effectiveness of our approach. Instead, we propose not to modify the regions of the scene but to arrange the features symmetrically.

Let us consider a voxel vertex $\mathbf{v}$ in a linear voxel grid (Fig. 3a), located at the boundary of regions $\mathbf{e}_1$ and $\mathbf{e}_2$. Note that in this case, the feature that will be seen by decoder θ_1 and decoder θ_2 at $\mathbf{v}$ will not be the same, discouraging the model from producing coherent results at both sides of the seam. In contrast, if the features are arranged symmetrically w.r.t. the scene center (Fig. 3b), we force the feature seen by any decoder at any boundary to be the same as the feature seen by all its neighboring decoders.

In practice, we also initialize all the density decoders with the same weights to improve continuity during training. The symmetric configuration also increases the probability of a feature being on empty space for all the regions of the volume. Therefore, this approach not only avoids increasing the number of features needed for a seamless reconstruction, but it also reduces the number of features that have to be stored after the pruning stage, further reducing the model size (see again Fig. 3). In Sec. 5.3, we will show how the use of a symmetric voxel grid turns out to be very effective to improve the model quality and reduce its size.

4.3. Deterministic Volume Integration

We apply deterministic integration to improve the quality of our model. Intersecting a ray with the voxel grid results in a series of intervals corresponding to the different voxels crossed by the ray. Instead of randomly sampling a point and then applying trilinear interpolation of the features of



(a) Linear voxel grid configuration.

(b) Symmetric voxel grid configuration

Figure 3. **Symmetric Voxel Grid configuration.** Arranging the features in a symmetric configuration within the voxel grid favours the continuity between regions by forcing the different decoders to share the same features at the boundaries. It also reduces the number of features needed to represent the scene, reducing the model size.

the given voxel to obtain the input feature, the trilinear interpolation operation is integrated along each of the intervals:

$$(\sigma_j, h_j) = \theta_i \left(\int_{t_j^{\text{in}}}^{t_j^{\text{out}}} \hat{\mathbf{f}}(\mathbf{x_j}) dt, \mathbf{d} \right), \quad i \in [1, 8], \tag{6a}$$

$$\mathbf{c}_j = \theta_{color}(h_j, \gamma(\mathbf{d}_j)), \tag{6b}$$

where t_j^{in} is the ray entry point of each voxel and t_j^{out} is the ray exit point of each voxel. This deterministic feature integration produces a better estimation for Eq. (1) compared with Monte Carlo since the density and color fields are not considered to be constant between samples. Instead, the whole interval is represented by the integration of the features belonging to the voxel where the given interval is defined.

4.4. Architecture and model training

The density decoders are composed of an input layer and a hidden layer of 32 neurons and an output layer of 33 neurons, while the color decoder consists only of an input layer of 32 neurons and an output layer of three neurons. All layers are activated with ReLU except for output layers, which have no activation in the case of density decoders and use sigmoid for the color decoder. For the implicit MLP, we use an architecture consisting of one input layer and eight hidden layers with 512 neurons, and an output layer with 32 neurons. All layers use ReLU activation except for the last one, where no activation is used. The positional encoding of the vertex coordinates is concatenated to the output of the fourth hidden layer.

Before training our model, we perform coarse geometry searching using a low-resolution voxel grid with a single region and one single-density decoder. We train this coarse model for five epochs and extract an occupancy mask, discarding the weights and the features of the model. This occupancy mask is obtained in a completely unsupervised way, i.e. it is extracted from the density outputs of the model. We use this mask to speed up our training by avoiding optimizing features located in empty space.

As discussed in [48], deterministic integration is prone to overfit at certain voxels when features are explicitly trained (i.e. directly optimized). To overcome this issue, we adopt the same hybrid regularization of the features proposed by [48]. This hybrid approach is composed of two stages: in the first one, features are implicitly optimized (i.e. an MLP is trained to obtain the features given the vertex coordinates). In the second one, the implicit MLP is discarded and features are explicitly optimized, leading to a better final quality. In this way, the MLP used at the implicit stage provides a powerful initialization that avoids overfitting.

To optimize our model, besides the photometric loss, we apply the sparsity regularization loss of [10]:

$$L_{\text{sparsity}} = \lambda_s \sum_i \log \left(1 + \frac{\sigma_i^2}{0.5} \right), \tag{7}$$

where σ_i represents the ith accumulated density and λ_s is the regularization weight. This regularization favors the sparsity of the voxel grid and prevents the model from predicting background color in empty space.

4.5. Model sparsification and real-time rendering

We leverage the sparsity of the voxel grid to store only the indices and values for the features of the occupied voxels. We also store a binary occupancy mask and the weights for the density and color decoders. To further optimize the model size, we follow the same strategy as Instant-NGP [26], and store the parameters using half precision.

At inference, we make use of an octree generated from the occupancy mask to avoid querying empty or occluded voxels, as in [54]. We do that by rendering the alpha maps for all training views and recording the maximum blended alpha for each voxel. Then, we prune the voxels with an



Figure 4. **Qualitative results** show that NeRFLight is able to model fine details at a similar quality than current state of the art models while achieving a 6× increment in the frame rate vs storage ratio of Instant-NGP and DIVeR.

alpha value below a threshold τ_α. While rendering a ray, we stop when the accumulated transmittance is below a certain threshold τ_t. If the alpha value of a given interval, after decoding the density value, is below τ_t, we do not evaluate the color decoder in that interval.

5. Experiments

We evaluate NeRFLight and compare it with the current state-of-the-art methods on the two most established benchmarks for bounded 360º scenes: the NeRF-synthetic dataset [25], consisting of 800 × 800 images of eight synthetic scenes and a subset of the Tanks and Temples dataset [15], provided in [20], composed of five real scenes of 1920 × 1080 images where the background has been masked. The NeRF-synthetic dataset allows testing the method on scenes with complex geometry and non-Lambertian materials. On the other side, using the Tanks and Temples dataset we can evaluate our model in real-world scenes. We also analyze the contribution of the training strategy and the symmetric grid configuration on an ablation study performed on the NeRF-synthetic dataset.

5.1. Implementation details

We implement our method using Pytorch [29] alongside custom CUDA kernels for the ray-voxel intersection. At inference, decoders evaluation and alpha blending are also implemented using CUDA kernels. The voxel grid resolution is set to 256^3 for the NeRF-synthetic dataset (256 voxels per side of the cubic voxel grid) and 320^3 for the Tanks and Temples dataset. As detailed in Sec. 4.1, the number of voxels for each of the eight regions are 128^3 and 160^3 respectively. The resolution for the voxel grid at the coarse stage is 1/4 of the resolution at the fine stage, and we also downsample training images to 1/4 of the original resolution. We set the sparsity regularization weight to $\lambda_s = 1e^{-5}$ an the alpha threshold for voxel pruning to $\tau_\alpha = 0.01$. We prune twice, after the coarse stage and after the fine stage. For rendering, we set τ_t also to 0.01.

At training, we sample rays from the training set using a batch size of 16384 for the NeRF-synthetic dataset and 8192 for Tanks and Temples. After the coarse geometry searching, we train NeRFLight with the implicit MLP for 1000 epochs, and, then, we train using the explicit grid for other 1000 epochs. We use Adam [14] as optimizer with a learning rate of 1e-3 for the coarse geometry searching and 5e-4 for the fine stage. For our experiments, we use an NVIDIA A100 GPU during training, and the peak memory usage is 40 GB approximately. At inference, we use an NVIDIA RTX 3090 GPU, except for the method marked as * in Tab. 1, whose metrics are the ones reported in [10] using an NVIDIA V100 GPU. We did not re-run these methods due to their extremely low FPS/MB metric.

5.2. Baseline Comparisons

Tab. 1 shows an exhaustive quantitative comparison of NeRFLight and current state-of-the-art methods. The quality metrics shown for the methods marked with * for the NeRF Synthetic dataset are the ones reported in [10] while, for the Tanks and Temples dataset, are the ones reported in [32]. The rest of the quality metrics are the ones reported in the original works.

NeRF Synthetic Dataset [25]						
Method	PSNR↑	SSIM↑	LPIPS↓	MB↓	FPS↑	FPS/MB↑
NeRF* [25]	31.00	0.947	0.081	5	0.03	0.006
JAXNeRF+* [7]	33.00	0.962	0.038	18	0.01	0.0006
JAXNeRF* [7]	31.65	0.952	0.051	*4.8*	0.05	0.0104
AutoInt* [19]	25.55	0.911	0.170	5	0.38	0.076
SNeRG(PNG) [10]	30.38	0.950	0.050	84	*202*	2.404
SNeRG(H264) [10]	29.86	0.938	0.065	30.2	*202*	*6.689*
Eff-NeRF [11]	31.68	0.954	**0.020**	648	**238**	0.367
KiloNeRF [32]	31.00	0.950	*0.030*	161	79	0.490
Plenoctrees [54]	31.71	0.958	0.053	1930	168	0.087
TensoRF-CP [6]	31.56	0.949	0.076	**3.9**	0.45	0.115
TensoRF-VM-192-30k [6]	*33.14*	0.963	0.047	71.8	0.38	0.0053
Instant-NGP [26]	**33.18**	**0.974**	0.043	27	62	2.296
DIVeR [48]	32.12	0.958	0.033	68	133	1.956
NeRFLight	31.41	*0.968*	0.039	14	181	**12.929**
Tanks and Temples Dataset [15]						
Method	PSNR↑	SSIM↑	LPIPS↓	MB↓	FPS↑	FPS/MB↑
NeRF* [25]	28.32	0.900	0.11	*5*	-	-
KiloNeRF [32]	*28.41*	0.910	0.090	-	41.3	-
Plenoctrees [54]	27.99	0.917	0.131	2629	42.22	0.016
TensoRF-CP [6]	27.59	0.897	0.181	**4.23**	0.27	0.064
TensoRF-VM-192-30k [6]	**28.56**	0.920	0.140	71.4	0.16	0.0022
Instant-NGP [26]	27.50	*0.935*	**0.076**	40	29	*0.725*
DIVeR [48]	28.21	0.906	*0.082*	116	*54*	0.47
NeRFLight	27.85	**0.939**	0.084	40	**78**	**1.95**

Table 1. **Quantitative results** on NeRF Synthetic and Tanks and Temples shows how NeRFLight is the method with the higher frame rate vs storage ratio while achieving competitive quality metrics. In **bold** we highlight the best performing model and the *underlined results* indicate the second best. * indicates the metrics are reported in a NVIDIA V100 GPU.

NeRF Synthetic. The results in the NeRF synthetic dataset show that current NeRF methods are not able to achieve a frame rate vs storage ratio higher than 2.5 FPS/MB with high-quality metrics. Whereas SNeRG(H264) [10] is able to achieve 6.689 FPS/MB thanks to the texture atlas compression, its quality metrics are substantially lower than models like Instant-NGP [26] or TensoRF [6]. In contrast, NeRFLight achieves 12.929 FPS/MB, almost 2× the frame rate vs storage ratio of SNeRG(H264) and more than 5× the ratio of Instant-NGP or DIVeR [48] with comparable quality metrics, achieving the second best result in SSIM. Fig. 4 shows qualitatively that NeRFLight is able to render complex fine structures at the same level of detail as the best state-of-the-art models.

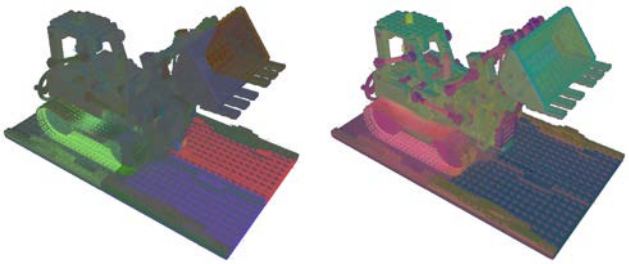
Tanks and Temples. Comparisons on the Tanks and Temples dataset show that NeRFLight is not only the method with the highest FPS/MB metric but also the most efficient at inference. Furthermore, the quality metrics demonstrate that NeRFLight is able to achieve accurate reconstruction, obtaining the best SSIM result. Fig. 4 shows that NeRFLight is also able to model fine high-frequency details even when applied to real-world scenes.

Instant-NGP [26] reports no results on this dataset, so we used the implementation provided by the original work to train the models for this experiment. Since Instant-NGP is highly configurable, we tune its hyperparameters so that the model size matches the size of NeRFLight. In this way, we can compare NeRFLight with the most accurate model in the current state of the art at the same model size. It can be seen how NeRFLight is able to outperform Instant-NGP in PSNR and SSIM while being able to render more than 2× faster. See supplemental materials for more details about Instant-NGP hyperparameters.

5.3. Ablation Study

We next conduct an ablation study of NeRFLight through two experiments that analyze the contribution of the symmetric grid and the implicit initialization of the features. First, we analyze the quality and continuity of the intermediate representations obtained by the density decoders using the Lego Bulldozer scene as an example. Then, we show quantitative and qualitative results of the different configurations of NeRFLight.

Decoders' continuity. In this experiment, we analyze the contribution of the symmetric voxel grid by inspecting the continuity of the intermediate representations obtained by the density decoders (h_j in Eq. (5b)). To do this, we apply alpha-composition of the intermediate representation at each point and apply PCA to reduce their dimension from



(a) Linear grid (b) Symmetric grid

Figure 5. **PCA decomposition** of density decoder's outputs shows that a symmetric grid is able to remove seams while a linear grid produces discontinuities.

Grid	Reg.	PSNR ↑	SSIM ↑	LPIPS ↓	FPS/MB ↑
Linear	Im	29.82	0.947	0.050	172/24=7.17
	Im-Ex	*30.98*	0.956	*0.046*	172/24=7.17
Symm	Im	30.56	*0.959*	*0.046*	**181/14=12.9**
	Im-Ex	**31.41**	**0.968**	**0.039**	**181/14=12.9**

Table 2. **Ablation of NeRFLight** shows that symmetric voxel grid configuration and implicit feature initialization not only improve the quality of the representation, but also improve the frame rate vs storage ratio.

32 to 3, so they can be rendered as an image. Fig. 5 shows the results on the Lego scene. It can be seen how applying a symmetric grid produces not only a seamless representation but also a better-clustered feature space. On the other side, applying a linear grid produces discontinuities, so the color decoder will have to struggle with this inconsistent representation. To demonstrate that our strategy is also robust to non-symmetric scenes, we provide another version of Fig. 5 in a rotated Lego scene in the supplemental materials.

Quality of the reconstruction. The artifacts produced at the frontiers between neighboring regions when applying a linear non-symmetric grid can be clearly seen in Fig. 6a. Fig. 6b shows that a symmetric grid is able to overcome this problem by producing smooth seamless transitions. Tab. 2 reports the quality and FPS/MB metrics for the different configurations of NeRFLight. These results demonstrate that applying a symmetric grid not only improves the quality of the reconstruction but also helps to achieve a higher frame rate vs storage ratio by reducing the number of features. This leads to a lighter model and faster access to the features for each voxel (faster rendering). Finally, Tab. 2 also shows that by combining implicit and explicit training, the quality of the final model is enhanced. We also provide some results applying direct optimization of the features (i.e. fully explicit regularization) in the supplemental materials, where we show that our method also achieves high-quality results using this training method.



(a) Linear grid (b) Symmetric grid

Figure 6. **Continuity at seams.** Locating the features symmetrically to the scene center in the voxel grid removes discontinuities at the seams between regions.

6. Limitations and future work

The most important limitation of NeRFLight is the amount of training time it requires, which is much larger than other methods like TensoRF [6] or Instant-NGP [26]. However, we believe that applying tensor decomposition as in TensoRF would reduce the convergence time of NeRFLight. Another limitation of our current implementation is that it is not able to represent unbounded or forward-facing scenes. In future work, we aim to address these other types of scenes by applying normalized device coordinates (NDC) or multi-sphere images (MSI). Finally, we believe that related research approaches might benefit from applying shared feature grids to their grid-based representations.

7. Conclusions

In this work, we have presented NeRFLight, the method with the highest frame rate vs storage cost ratio of the state of the art. We show how, even when optimizing for reduced size and inference efficiency, our method provides very good rendering quality results, on par with the most recent NeRF models. We also introduced the concept of a shared feature grid in NeRF, as a way to reuse features to make grid representations more compact. Furthermore, we have proposed the first approach to achieve seamless reconstructions using symmetric voxel grids. We have shown that this strategy, reduces the number of features in the grid and leads to a faster rendering and a more accurate reconstruction. To summarize, NeRFLight is the current NeRF-based method with the best FPS/MB ratio, which makes it especially adequate to stream and render in scenarios like social media, online video games, or for upcoming metaverse applications.

Acknowledgements

This work is partially supported by the Spanish government under project MoHuCo PID2020-120049RB-I00.

References

[1] Edward H. Adelson and James R. Bergen. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, pages 3–20. MIT Press, 1991. 1, 2

[2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *Computer Vision – ECCV 2020*, pages 696–712, 2020. 1, 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 (CVPR)*, pages 5470–5479, June 2022. 2

[4] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. Nerd: Neural reflectance decomposition from image collections. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12664–12674, 2021. 2

[5] Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan Barron, and Hendrik PA Lensch. Neural-pil: Neural pre-integrated lighting for reflectance decomposition. In *Advances in Neural Information Processing Systems*, volume 34, pages 10691–10704, 2021. 2

[6] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 3, 7, 8

[7] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an efficient JAX implementation of NeRF, 2020. 7

[8] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14304–14314, 2021. 2

[9] 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 (CVPR)*, pages 5501–5510, June 2022. 2

[10] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5855–5864, 2021. 2, 5, 6, 7

[11] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. Efficientnerf efficient neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12902–12911, June 2022. 2, 7

[12] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18(3):165–174, jan 1984. 3

[13] Petr Kellnhofer, Lars C. Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4285–4295, 2021. 2

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 6

[15] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4), jul 2017. 6, 7

[16] Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas Funkhouser. Panoptic Neural Fields: A Semantic Object-Aware Neural Scene Representation. In *CVPR*, 2022. 2

[17] Tianye Li, Mira Slavcheva, Michael Zollhöfer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5521–5531, June 2022. 2

[18] 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 (CVPR)*, pages 6498–6508, June 2021. 2

[19] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14556–14565, June 2021. 7

[20] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 3, 6

[21] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. Editing conditional radiance fields. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 2

[22] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 38(4), jul 2019. 1, 2

[23] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14214–14223, October 2021. 4

[24] 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 Trans. Graph.*, 38(4), jul 2019. 2

[25] 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, 4, 6, 7

[26] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multires-

olution hash encoding. *ACM Trans. Graph.*, 41(4), jul 2022. 2, 3, 5, 7, 8

[27] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3501–3512, 2020. 2

[28] 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 *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5845–5854, 2021. 2

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of Advances in Neural Information Processing Systems*, volume 32, 2019. 6

[30] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10313–10322, 2021. 2

[31] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14153–14161, June 2021. 3

[32] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14315–14325, 2021. 2, 3, 6, 7

[33] Konstantinos Rematas and Vittorio Ferrari. Neural voxel renderer: Learning an accurate and controllable rendering tool. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5416–5426, 2020. 2

[34] Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Tom Funkhouser, and Vittorio Ferrari. Urban radiance fields. *CVPR*, 2022. 2

[35] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *Computer Vision – ECCV 2020*, pages 623–640, 2020. 2

[36] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12216–12225, June 2021. 2

[37] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Hao Li, and Angjoo Kanazawa. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2304–2314, 2019. 2

[38] 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 (CVPR)*, June 2019. 2

[39] Vincent Sitzmann, Michael Zollhoefer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 2

[40] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7491–7500, 2021. 2

[41] Pratul P. Srinivasan, Richard Tucker, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2

[42] 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 (CVPR)*, pages 5459–5469, June 2022. 2

[43] Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*. Association for Computing Machinery, 2022. 3

[44] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-NeRF: Scalable large scene neural view synthesis. *arXiv*, 2022. 2

[45] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 38(4), jul 2019. 2

[46] Cen Wang, Minye Wu, Ziyu Wang, Liao Wang, Hao Sheng, and Jingyi Yu. Neural opacity point cloud. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(7):1570–1581, 2020. 2

[47] 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

[48] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David Forsyth. Diver: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16200–16209, June 2022. 2, 3, 5, 7

[49] 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 (CVPR)*, pages 9421–9431, 2021. 2

[50] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *The European Conference on Computer Vision (ECCV)*, 2022. 2

[51] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *Computer Graphics Forum*, 2022. 2

[52] Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Learning object-compositional neural radiance field for editable scene rendering. In *International Conference on Computer Vision (ICCV)*, October 2021. 2

[53] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2492–2502. Curran Associates, Inc., 2020. 2

[54] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5732–5741, 2021. 2, 5, 7

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

[56] Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul Debevec, William T. Freeman, and Jonathan T. Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *ACM Trans. Graph.*, 40(6), dec 2021. 2

[57] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Trans. Graph.*, 37(4), jul 2018. 2

NeRFLight Supplemental Materials

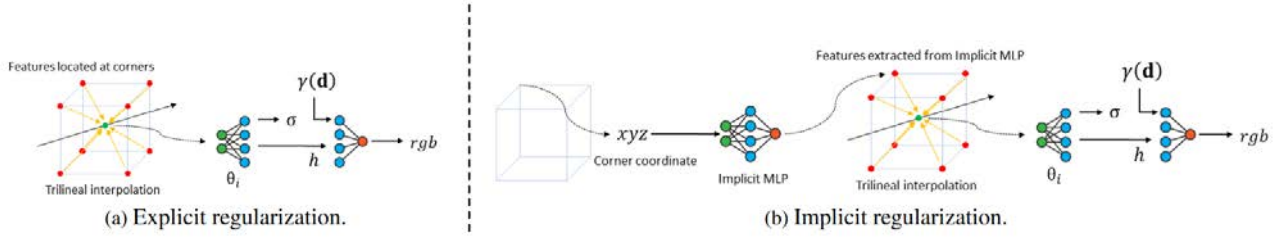


Figure 7. **Grid regularizations** Features can be obtained either using an auto-decoder architecture (a) where the features are directly optimized or using a auto-encoder architecture (b) where an implicit MLP is optimized to obtained the features depending on their spatial position.

In this section, we provide additional details about the architecture of the MLPs used in our work together with a more detailed description of the feature optimization methods applied (explicit and implicit regularization). We also report more detailed per-scene quantitative results and additional qualitative results.

A. Additional MLP architecture details

Fig. 7 shows the two different training strategies that are used to train NeRFLight. In the coarse stage, features are initialized randomly and optimized using explicit regularization. However, at the fine stage, the features are first optimized using an auto-encoder architecture, where an implicit MLP is used to obtain a feature value depending on its spatial location. After 1000 epochs, the features for each of the corners in the grid are extracted. These values are used as initialization for an auto-decoder architecture, where features are, once again, directly optimized.

Fig. 8 shows the architecture of the implicit MLP used for implicit regularization. It is composed of one input layer and eight hidden layers of 512 neurons activated using ReLU and an output layer of 32 neurons without activation. Positional encoding is applied to the corner coordinates. We use ten frequencies for this positional encoding.

Fig. 9 shows the architecture of a density decoder and a color decoder. Each of the density decoders is composed of one input layer, one hidden layer, and one output layer. They take the integrated feature (dimension 32) as input and output the density value σ and an intermediate representation h. Each of the layers is activated with ReLU activation except for the neurons of the last layer that correspond to the intermediate representation, where no activation is used. This intermediate representation is concatenated with the positional encoding of the by direction to produce the input of the color decoder. Finally, the color decoder, composed of a ReLU-activated input layer of 32 neurons and a sigmoid-activated output layer of 3 neurons, obtains the color of the given interval. Four frequencies are used for the direction positional encoding.

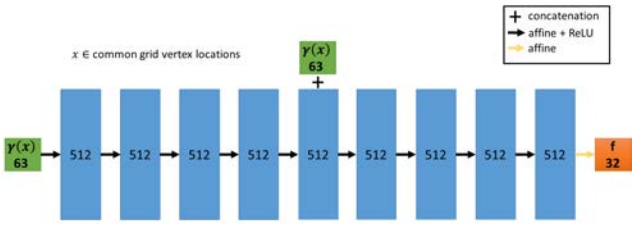


Figure 8. **Implicit MLP architecture.** The implicit MLP is used to obtain a robust initialization of the feture grid. It is evaluated on the corners of the common feature grid.

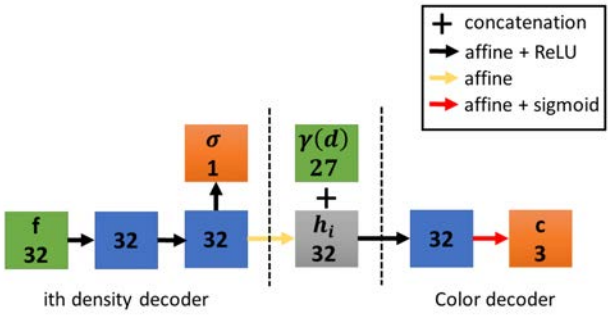
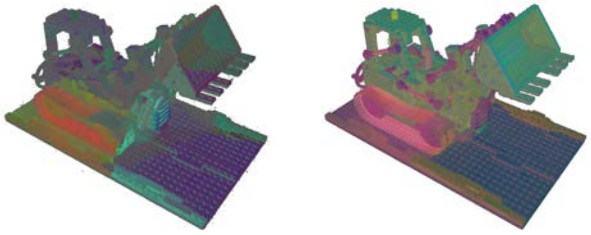


Figure 9. **Density and color decoders.** Depending on the location of the voxel, one of the N density decoders obtains the density value at the given voxel and an intermediate representation from the integrated feature. Then, the color decoder used the intermediate representation and the direction positional encoding to obtain the color value.



(a) Rotated Linear grid (b) Rotated Symmetric grid

Figure 10. **PCA decomposition** of density decoder's outputs of the rotated Lego Scene. The symmetric grid shows to be more stable.

B. Additional Results

Fig. 10 shows the same experiment as Fig. 5 but using a Lego scene rotated $(14°, 28°, 64°)$ on the X-Y-Z axes. The goal of this experiment is to show that our approach is also robust for the case where the symmetry axes of the scenes are not aligned with the axes of the scene (as occurs in some scenes of the NeRF Synthetic dataset. Tab. 3 shows that the metrics obtained by our model in both scenes (original and rotated) are similar. Tab. 4 shows the results on Lego Scene from the NeRF Synthetic dataset of NeRFLight using different regularizations. It can be seen how all of them (explicit, implicit, and hybrid) obtain high-quality results. While explicit regularization obtains the lowest quality metrics, it requires less time and GPU memory. For the lego scene, the training time was 16 hours and the peak GPU memory usage was 22GB, a significant reduction compared with the 60 hours and the 40GB needed by the hybrid regularization.

In Tab. 5, we show a per-scene comparison of the rendering quality between NeRFLight and all the baselines we compared with in NeRF synthetic dataset. In Tab. 6, we do the same for the Tanks and Temples dataset. We also provide the model size and frame rate for NeRF synthetic dataset (Tab. 7) and the model size for the Tanks and Temples dataset Tab. 8. In these results, we also add the DIVeR-64 model [48] for NeRF Synthetic dataset, while the tables shown in the paper only reports the results from DIVeR-32 since it is the model of [48] with the best FPS/MB ratio, while the quality metrics are really similar between both models.

As explained in the paper, in Tab. 1 we provide the results of Instant-NGP for Tanks and Temples dataset tuning its hyperparameters so that the model size matches NeRFlight. We use a hash table size of 2^{22} and the finest resolution is set to 256. In Tab. 6 also shows the results of a larger model of Instant-NGP using the same size for the hash table and the finest resolution of 512. It is marked as (l) while the Instant-NGP shown in the paper is marked as (s).

Finally, we also provide additional qualitative results in Fig. 12 and Fig. 11.

Orientation	PSNR ↑	SSIM ↑	LPIPS ↓
Axis aligned	31.41	0.968	0.039
Rotated	31.32	0.967	0.039

Table 3. Results on Lego scene with and w/o rotation.

Reg.	PSNR ↑	SSIM ↑	LPIPS ↓
Ex	33.02	0.975	0.016
Im	33.24	0.979	0.013
Im-Ex	**33.84**	**0.985**	**0.011**

Table 4. Results on Lego scene using different regularizations

	PSNR ↑								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF*	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65	31.00
JAXNeRF+*	35.35	25.65	32.77	37.58	35.35	30.29	36.52	30.48	33.00
JAXNeRF*	33.88	25.08	30.51	36.91	33.24	30.03	34.52	29.07	31.65
AutoInt*	25.60	20.78	22.47	32.33	25.09	25.90	28.10	24.15	25.55
SNeRG(PNG)	33.24	24.57	29.32	34.33	33.82	27.21	32.60	27.97	30.38
SNeRG(H264)	-	-	-	-	-	-	-	-	29.86
Eff-NeRF	-	-	-	-	-	-	-	-	31.68
KiloNeRF	32.91	25.25	29.76	35.56	33.02	29.20	33.06	29.23	31.00
Plenoctrees	34.66	25.31	30.79	36.79	32.95	29.76	33.97	29.42	31.71
TensorRF-CP	33.60	25.17	30.72	36.24	34.05	30.10	33.77	28.84	31.56
TensorRF-VM	35.76	26.01	33.99	37.41	36.46	30.12	34.51	30.77	*33.14*
Instant-NGP	35.00	26.02	33.51	37.40	36.39	29.78	36.22	31.10	**33.18**
DIVeR-64	34.35	25.38	31.76	36.76	35.49	29.61	34.57	30.48	32.30
DIVeR-32	34.09	25.40	32.02	36.35	35.17	29.24	34.53	30.14	32.12
NeRFLight	32.76	25.60	31.65	35.96	33.84	29.00	33.1	29.38	31.41

	SSIM ↑								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF*	0.967	0.925	0.964	0.974	0.961	0.952	0.987	0.865	0.947
JAXNeRF+*	0.982	0.936	0.980	0.983	0.979	0.956	0.991	0.887	0.962
JAXNeRF*	0.974	0.927	0.967	0.979	0.968	0.952	0.987	0.865	0.952
AutoInt*	0.928	0.861	0.898	0.974	0.900	0.930	0.948	0.852	0.911
SNeRG(PNG)	0.975	0.929	0.967	0.971	0.973	0.938	0.982	0.865	0.950
SNeRG(H264)	-	-	-	-	-	-	-	-	0.938
Eff-NeRF	-	-	-	-	-	-	-	-	0.954
KiloNeRF	0.970	0.930	0.970	0.980	0.9700	0.950	0.980	0.880	0.950
Plenoctrees	0.981	0.933	0.970	0.982	0.971	0.955	0.987	0.884	0.958
TensorRF-CP	0.973	0.921	0.965	0.975	0.971	0.950	0.983	0.857	0.949
TensorRF-VM	0.985	0.937	0.982	0.982	0.983	0.952	0.988	0.895	0.963
Instant-NGP	0.987	0.950	0.987	0.989	0.988	0.964	0.991	0.938	**0.974**
DIVeR-64	-	-	-	-	-	-	-	-	-
DIVeR-32	0.977	0.932	0.977	0.978	0.978	0.946	0.987	0.885	0.958
NeRFLight	0.978	0.954	0.983	0.990	0.985	0.949	0.989	0.927	*0.968*

	LPIPS ↓								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF*	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081
JAXNeRF+*	0.017	0.057	0.018	0.022	0.017	0.041	0.008	0.123	0.038
JAXNeRF*	0.027	0.070	0.033	0.030	0.030	0.048	0.013	0.156	0.051
AutoInt*	0.141	0.224	0.148	0.080	0.175	0.136	0.131	0.323	0.170
SNeRG(PNG)	0.025	0.061	0.028	0.043	0.022	0.052	0.016	0.156	0.050
SNeRG(H264)	-	-	-	-	-	-	-	-	0.065
Eff-NeRF	-	-	-	-	-	-	-	-	**0.020**
KiloNeRF	0.020	0.050	0.020	0.020	0.020	0.020	0.010	0.080	*0.030*
Plenoctrees	0.022	0.076	0.038	0.032	0.034	0.059	0.017	0.144	0.053
TensorRF-CP	0.044	0.114	0.058	0.052	0.038	0.068	0.035	0.196	0.076
TensorRF-VM	0.022	0.073	0.022	0.032	0.018	0.058	0.015	0.138	0.047
Instant-NGP	0.012	0.08	0.053	0.013	0.014	0.063	0.023	0.086	0.043
DIVeR-64	-	-	-	-	-	-	-	-	-
DIVeR-32	0.014	0.058	0.020	0.019	0.010	0.034	0.011	0.100	0.033
NeRFLight	0.025	0.073	0.036	0.010	0.011	0.045	0.030	0.08	0.039

Table 5. **Rendering quality on NeRF synthetic dataset**

	PSNR ↑					
	Barn	Caterpillar	Family	Ignatius	Truck	Mean
NeRF	27.71	25.87	33.33	27.79	26.92	28.32
KiloNeRF	27.81	25.61	33.65	27.92	27.04	28.41
Plenoctrees	26.80	25.29	32.85	28.19	26.83	27.99
TensoRF-CP	26.74	24.73	32.39	27.86	26.25	27.59
TensoRF-VM	27.22	26.19	33.92	28.34	27.14	*28.56*
Instant-NGP(s)	26.51	24.9	32.98	27.63	25.49	27.50
Instant-NGP(l)	27.41	26.61	34.68	28.23	27.92	**28.97**
DIVeR-32	26.7	25.8	33.34	27.42	27.81	28.21
NeRFLight	26.42	25.3	33.23	27.24	27.22	27.85

	SSIM ↑					
	Barn	Caterpillar	Family	Ignatius	Truck	Mean
NeRF	0.85	0.89	0.95	0.94	0.89	0.90
KiloNeRF	0.85	0.90	0.96	0.94	0.90	0.910
Plenoctrees	0.856	0.907	0.962	0.948	0.914	0.917
TensoRF-CP	0.839	0.879	0.948	0.934	0.885	0.897
TensoRF-VM	0.864	0.912	0.965	0.948	0.914	0.920
Instant-NGP(s)	0.897	0.924	0.968	0.963	0.921	0.935
Instant-NGP(l)	0.910	0.940	0.980	0.970	0.942	**0.948**
DIVeR-32	0.843	0.900	0.954	0.932	0.900	0.906
NeRFLight	0.891	0.927	0.976	0.958	0.941	*0.939*

	LPIPS ↓					
	Barn	Caterpillar	Family	Ignatius	Truck	Mean
NeRF	0.19	0.12	0.05	0.08	0.11	0.11
KiloNeRF	0.16	0.10	0.04	0.06	0.10	0.090
Plenoctrees	0.226	0.148	0.069	0.080	0.130	0.131
TensoRF-CP	0.237	0.176	0.063	0.089	0.154	0.144
TensoRF-VM	0.217	0.139	0.057	0.081	0.129	0.125
Instant-NGP(s)	0.151	0.056	0.036	0.075	0.060	*0.076*
Instant-NGP(l)	0.142	0.051	0.034	0.064	0.053	**0.069**
DIVeR-32	0.142	0.095	0.041	0.054	0.079	0.082
NeRFLight	0.139	0.100	0.043	0.055	0.083	0.084

Table 6. **Rendering quality on the Tanks and Temples dataset**

	MB↓								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF*	5	5	5	5	5	5	5	5	5
JAXNeRF+*	18	18	18	18	18	18	18	18	18
JAXNeRF*	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	*4.8*
AutoInt*	5	5	5	5	5	5	5	5	5
SNeRG(PNG)	-	-	-	-	-	-	-	-	84
SNeRG(H264)	-	-	-	-	-	-	-	-	30.2
Eff-NeRF	-	-	-	-	-	-	-	-	648
KiloNeRF	204	-	-	-	108	-	-	173	161
Plenoctrees	832	1239	1792	2683	2068	3686	443	2693	1930
TensorRF-CP	3.71	3.78	3.73	4.23	3.74	4.49	3.83	3.87	**3.9**
TensorRF-VM	66.7	69.9	68.6	83.8	81.5	67.8	70.2	68.1	71.8
Instant-NGP	27	27	27	27	27	27	27	27	27
DIVeR-64	55	42	49	80	64	62	24	118	62
DIVeR-32	55	56	47	84	64	62	24	151	68
NeRFLight	9	15	9	16	13	11	10	30	14

	FPS↑								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF*	-	-	-	-	-	-	-	-	0.03
JAXNeRF+*	-	-	-	-	-	-	-	-	0.01
JAXNeRF*	-	-	-	-	-	-	-	-	0.05
AutoInt*	-	-	-	-	-	-	-	-	0.38
SNeRG(PNG)	-	-	-	-	-	-	-	-	*202*
SNeRG(H264)	-	-	-	-	-	-	-	-	*202*
Eff-NeRF	-	-	-	-	-	-	-	-	**238**
KiloNeRF	103	-	-	-	88	-	-	45	79
Plenoctrees	312	302	116	89	280	70	320	58	168
TensorRF-CP	0.66	0.752	0.15	0.21	0.24	0.42	0.69	0.49	0.45
TensorRF-VM	0.60	0.747	0.08	0.11	0.17	0.36	0.61	0.42	0.38
Instant-NGP	85	61	68	63	66	37	99	17	62
DIVeR-64	101	87	78	82	168	70	150	78	102
DIVeR-32	160	117	108	117	200	114	168	79	133
NeRFLight	223	252	149	130	215	121	201	153	181

	FPS/MB↑								
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NeRF*	-	-	-	-	-	-	-	-	0.006
JAXNeRF+*	-	-	-	-	-	-	-	-	0.0006
JAXNeRF*	-	-	-	-	-	-	-	-	0.0104
AutoInt*	-	-	-	-	-	-	-	-	0.076
SNeRG(PNG)	-	-	-	-	-	-	-	-	2.404
SNeRG(H264)	-	-	-	-	-	-	-	-	*6.689*
Eff-NeRF	-	-	-	-	-	-	-	-	0.367
KiloNeRF	0.505	-	-	-	0.815	-	-	0.260	0.490
Plenoctrees	0.375	0.244	0.065	0.033	0.135	0.019	0.722	0.022	0.085
TensorRF-CP	0.178	0.199	0.040	0.050	0.064	0.094	0.180	0.123	0.115
TensorRF-VM	0.009	0.011	0.0012	0.0013	0.002	0.005	0.009	0.006	0.0053
Instant-NGP	3.15	2.26	2.52	2.33	2.44	1.37	3.67	0.63	2.296
DIVeR-64	1.84	2.07	1.59	1.025	2.625	1.129	6.25	0.661	1.645
DIVeR-32	2.91	2.09	2.30	1.393	3.125	1.84	7.00	0.52	1.956
NeRFLight	24.78	16.8	16.56	8.125	16.54	11	20.1	5.1	**12.929**

Table 7. **Model size and rendering speed on NeRF synthetic dataset**

	MB↓					
	Barn	Caterpillar	Family	Ignatius	Truck	Mean
NeRF	5	5	5	5	5	*5*
KiloNeRF	-	-	-	-	-	-
Plenoctrees	2509	2365	2928	3031	2314	2629
TensorRF-CP	4.2	3.93	4.07	4.32	4.64	**4.23**
TensorRF-VM	71.5	67.4	66.7	80.7	70.6	71.4
Instant-NGP(s)	40	40	40	40	40	40
Instant-NGP(l)	116	116	116	116	116	116
DIVeR-32	85	92	133	154	117	116
NeRFLight	31	50	24	43	52	40

	FPS↑					
	Barn	Caterpillar	Family	Ignatius	Truck	Mean
NeRF*	-	-	-	-	-	0.005
KiloNeRF	-	-	-	-	-	41.3
Plenoctrees	23	37	58	51	42	42
TensorRF-CP	0.15	0.35	0.35	0.28	0.21	0.27
TensorRF-VM	0.02	0.24	0.22	0.18	0.16	0.16
Instant-NGP(s)	16	30	38	34	27	29
Instant-NGP(l)	16	30	38	34	27	29
DIVeR-32	31	56	70	66	48	*54*
NeRFLight	45	97	90	75	82	**78**

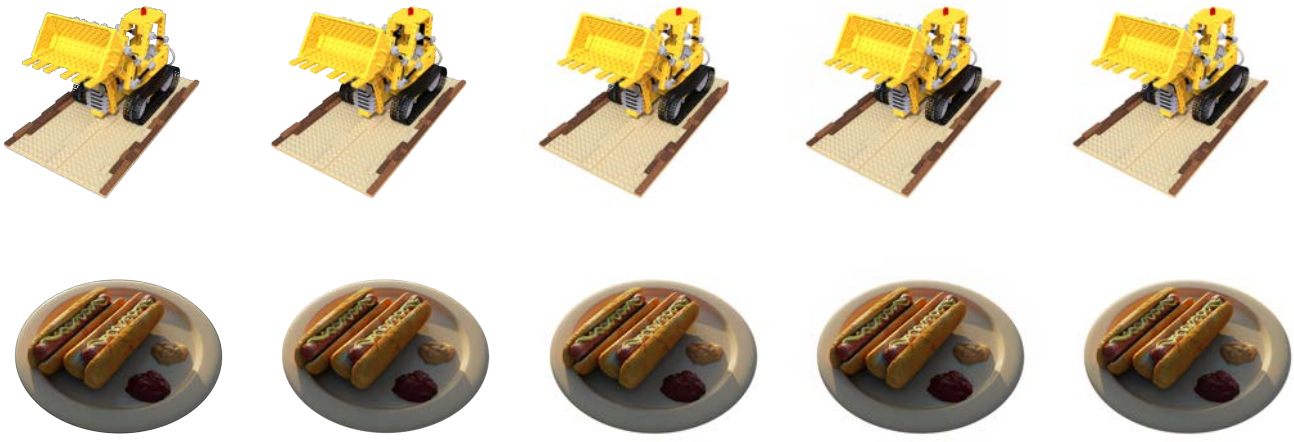
	FPS/MB↑					
	Barn	Caterpillar	Family	Ignatius	Truck	Mean
NeRF*	-	-	-	-	-	0.001
KiloNeRF	-	-	-	-	-	-
Plenoctrees	0.009	0.016	0.020	0.017	0.018	0.016
TensorRF-CP	0.036	0.089	0.086	0.065	0.045	0.064
TensorRF-VM	0.0003	0.004	0.003	0.002	0.002	0.0022
Instant-NGP(s)	0.4	0.75	0.95	0.85	0.675	*0.725*
Instant-NGP(l)	0.14	0.26	0.33	0.293	0.232	0.25
DIVeR-32	0.36	0.61	0.53	0.43	0.41	0.47
NeRFLight	1.45	1.94	3.75	1.74	1.58	**1.95**

Table 8. **Model size on Tanks and Temples dataset.**



Ground Truth Instant-NGP DIVeR TensorRF-VM NeRFLight

Figure 11. **Additional qualitative results on Tanks and Temples**.



Ground Truth Instant-NGP DIVeR TensorRF-VM NeRFLight

Figure 12. **Additional qualitative results on NeRF Synthetic**.