

# Re-ReND: Real-time Rendering of NeRFs across Devices

Sara Rojas<sup>1</sup>, Jesus Zarzar<sup>1</sup>, Juan C. Pérez<sup>1</sup>,  
 Artsiom Sanakoyeu<sup>2</sup>, Ali Thabet<sup>2</sup>, Albert Pumarola<sup>2</sup>, and Bernard Ghanem<sup>1</sup>

<sup>1</sup>King Abdullah University of Science and Technology (KAUST), <sup>2</sup>Meta Research

## Abstract

*This paper proposes a novel approach for rendering a pre-trained Neural Radiance Field (NeRF) in real-time on resource-constrained devices. We introduce Re-ReND, a method enabling Real-time Rendering of NeRFs across Devices. Re-ReND is designed to achieve real-time performance by converting the NeRF into a representation that can be efficiently processed by standard graphics pipelines. The proposed method distills the NeRF by extracting the learned density into a mesh, while the learned color information is factorized into a set of matrices that represent the scene’s light field. Factorization implies the field is queried via inexpensive MLP-free matrix multiplications, while using a light field allows rendering a pixel by querying the field a single time—as opposed to hundreds of queries when employing a radiance field. Since the proposed representation can be implemented using a fragment shader, it can be directly integrated with standard rasterization frameworks. Our flexible implementation can render a NeRF in real-time with low memory requirements and on a wide range of resource-constrained devices, including mobiles and AR/VR headsets. Notably, we find that Re-ReND can achieve over a 2.6-fold increase in rendering speed versus the state-of-the-art without perceptible losses in quality.*

## 1. Introduction

Neural Radiance Fields (NeRFs) [20] have revolutionized the field of novel view synthesis, as demonstrated by their impressive capacity to reconstruct complex objects and scenes with remarkable detail [2, 31]. The impressive performance of NeRFs casts them as a decisive tool for capturing and representing 3D objects and scenes. As a result, NeRFs hold great promise for countless practical applications, including video games, movies, and Augmented/Virtual Reality (AR/VR).

However, the impressive performance of NeRFs comes with significant computational costs when rendering novel views. This slow rendering is mainly due to two limit-



Figure 1. **Rendering NeRFs in real-time on resource-constrained devices, such as AR/VR headsets and mobiles.** We present Re-ReND, a method for rendering a pre-trained NeRF in real-time on a variety of devices with constrained computational resources. Re-ReND preserves remarkable photo-metric quality even when rendering at over 1,013 FPS on a desktop browser, or at the capped 74 FPS of a VR headset. Please, refer to the accompanying video for a demonstration of Re-ReND’s capacities.

ing properties of NeRFs, which dramatically increase their computational requirements. Firstly, they use a volumetric representation to model scenes [9], implying that rendering a single pixel requires hundreds of queries in space. Secondly, they leverage a Multilayer Perceptron (MLP) to model radiance and density across space, meaning that each of those spatial queries involves evaluating an expensive MLP. These properties make it challenging to render NeRFs in real-time on resource-constrained devices, which hinders their practical deployment.

A plethora of research efforts target the efficiency of NeRFs. A line of work focused on shortening training times [27, 8, 21], while another looked at accelerating rendering times [22, 26]. Notably, some methods pre-tabulate a NeRF’s output on a sparse grid, and achieve real-time rendering by leveraging powerful GPUs [36, 13]. However, these methods are still incompatible with widely-available graphics pipelines that enable rendering on resource-constrained devices via popular frameworks such as WebGL and OpenGL ES. This incompatibility stems

from the rendering approach inherited from NeRFs, as volume rendering demands ray marching, which is dramatically more expensive than mesh rasterization.

In this paper, we present Re-ReND, a method that enables Real-time Rendering of NeRFs across Devices, including resource-constrained devices such as VR headsets and mobile phones. To attain this end goal, we define and achieve three objectives: *(i)* enabling compatibility with widely-available graphics pipelines, *(ii)* obtaining ray color with a single query, and *(iii)* avoiding MLP evaluations for such queries. Given a pre-trained NeRF as input, Re-ReND renders it in real-time by transforming the knowledge learned by the NeRF into an alternative representation. In particular, Re-ReND distills the NeRF by extracting the learned density into a mesh, and the learned color information into a set of matrices that efficiently factorize the scene’s light field. Re-ReND is thus capable of rendering a NeRF in real-time, making it deployable on a wide range of devices. When rendering challenging unbounded real scenes, our method achieves over 2.6-fold speed improvements above the state-of-the-art, while maintaining comparable quality. Furthermore, given that our rendering entirely disposes of MLPs, we can easily deploy it on other constrained devices, like VR headsets, where other methods cannot adapt. Please refer to Figure 1 for an overview of the capabilities of Re-ReND.

In summary, our contributions are threefold: **(i)** We introduce Re-ReND, a method enabling real-time rendering of a pre-trained NeRF on resource-constrained devices. Re-ReND works on a wide variety of NeRFs, achieves remarkable rendering speeds at negligible costs to photometric quality, and is compatible with popular graphics pipelines available in common devices. **(ii)** Re-ReND enables fast rendering of a NeRF by transforming it into a representation with three fundamental qualities: it resembles a graphics-friendly representation (*i.e.* a mesh-texture package), it obtains ray color via light fields (avoiding expensive volume rendering), and it factorizes the light field computation as an MLP-free matrix multiplication. **(iii)** We conduct a comprehensive empirical study of Re-ReND across resource-constrained devices. Our results find remarkable boosts in rendering time that come at insignificant costs to image quality. Notably, we find Re-ReND boosts rendering speeds by  $2.6\times$  at low memory expenses in challenging real scenes, enabling real-time rendering on various devices.

Striving for reproducibility, we provide our implementation of Re-ReND, written in PyTorch [24], in the **Appendix**.

## 2. Related Work

**Light Field Representations.** Light fields [17] represent a scene as integrated radiance along rays, *i.e.* they are the integral of radiance fields. Unlike radiance fields, light fields directly map ray origin and direction to color, allowing for

a single query to compute a pixel’s color instead of multiple queries. Innovative works have aimed at parameterizing light fields with neural networks [28, 29, 1, 10, 32]. However, despite their desirable properties, light field representations have been found to be challenging to learn [16]. This difficulty mainly stems from *(i)* their reductive formulation of appearance requiring compensation with clever parameterizations [28, 10] or larger and data-hungry architectures [32], and *(ii)* their indirect use of geometry that calls for injecting geometric priors elsewhere in the pipeline [29, 1]. Our proposed Re-ReND renders a pre-trained NeRF in real time by distilling the NeRF’s color knowledge into a light field, thus leveraging the inherent one-query-per-pixel virtue of these fields. In particular, our light field formulation is similar to [35], whereby the field generates the color at the point of intersection between the camera ray and the scene’s geometry. We further accelerate rendering by factorizing the light field into matrices, and enabling compatibility with graphics pipelines by storing such matrices into texture map-like arrays.

**Factorizing Neural Fields.** NeRFs [20] use an MLP to map a position  $\mathbf{p} \in \mathbb{R}^3$  and a view direction  $\mathbf{d} \in \mathbb{R}^2$  to color and density. Various works have studied the complex interplay between position and view direction that happens inside NeRFs [13, 11, 36, 27, 34]. Notably, Garbin *et al.* [11] highlight how caching a NeRF’s output is precluded by the dependence of color on *both*  $\mathbf{p}$  and  $\mathbf{d}$ . Thus, the authors propose to factorize color into two independent functions (separately for  $\mathbf{p}$  and  $\mathbf{d}$ ), such that the inner product of their outputs generates color. This factorization allows for caching the NeRF’s output, and thus effectively disposes of MLPs for rendering. Our work enables real-time rendering by combining this factorization approach with the color formulation of light fields. Specifically, we leverage the NeRF-factorization method of [11] to factorize a Neural *Light Field*. This combination allows Re-ReND to enjoy both the factorization’s MLP-free rendering, and the one-query-per-pixel virtue of light fields.

**Rendering of Neural Fields.** Neural fields achieve impressive photo-realistic quality at undesirably large computational costs [20]. Various works have addressed training costs [21, 27, 5, 30], while others focused on rendering. For rendering, advances in differentiable and fast rendering [14, 23, 7] have enabled novel advances and applications [39]. While several approaches achieve real-time rendering on power-intensive setups [11, 36, 26, 18, 25] with access to GPUs, we focus on enabling real-time rendering on resource-constrained devices such as AR/VR headsets and mobiles. Rendering can be accelerated by exploiting graphics pipelines offered by the hardware of these devices. Both SNeRG [13] and PlenOctrees [36] leveraged optimized graphics routines for in-browser rendering, but did not exploit efficient mesh-oriented pipelines.

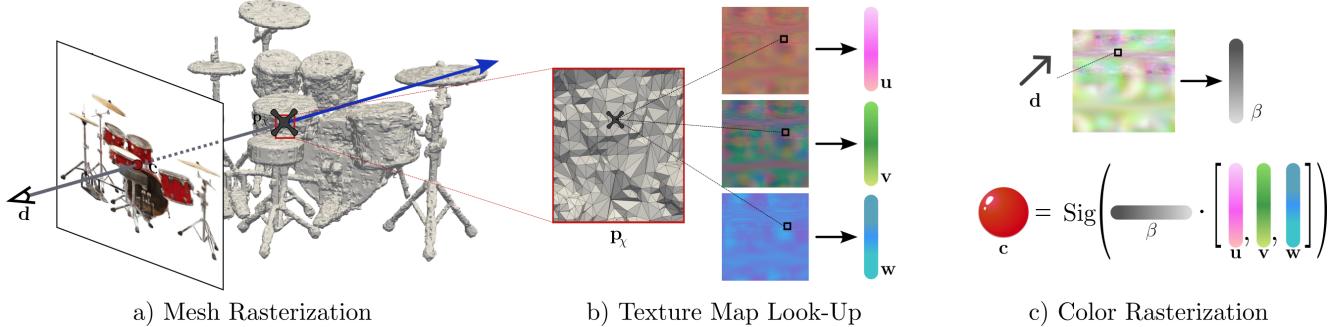


Figure 2. **Rendering a NeRF using Re-ReND.** Given a pre-trained NeRF, Re-ReND renders it in real time by leveraging a mesh and light field embedding maps ( $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$  and  $\beta$ ) which are compatible with a standard rasterization pipeline with programmable shaders. Rendering occurs in three steps: **a)** The mesh gets rasterized, and its vertices go through a regular vertex shader. **b)** The light field embedding maps are indexed (with the  $uv$  coordinates from the rasterization) to obtain values for the position-dependent ( $\mathbf{u}$ ,  $\mathbf{v}$  and  $\mathbf{w}$ ) and direction-dependent ( $\beta$ ) embeddings. **c)** Pixel colors are obtained by combining  $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$  and  $\beta$  in a custom fragment shader implementing an inexpensive dot product. Since Re-ReND uses light fields, alpha compositing is unnecessary, and so we set alpha values to 1. Note that our method entirely disposes of MLPs at render time, and thus enjoys substantial boosts in rendering speed.

These pipelines are difficult to exploit in NeRF’s formulation, since the volume rendering [9] nature of NeRFs is incompatible with the polygon-oriented paradigm of mesh rasterization. In this work, we accelerate the rendering of NeRFs by transforming their learned knowledge into a mesh-friendly representation on which efficient mesh-rasterization pipelines can operate. In particular, we represent a scene as a mesh whose texture maps are filled with “light field embeddings”, which result from factorizing the scene’s light field. Concurrently with our work, Chen *et al.* [6] propose MobileNeRF, which also exploits mesh-rasterization pipelines for fast rendering on devices. While MobileNeRF uses surface-based neural fields, Re-ReND uses light fields at the surface of objects. Furthermore, our method acts as a framework for *transforming* a pre-trained NeRF to an MLP-free and rasterization-friendly representation that is capable of real-time rendering across devices.

### 3. Problem Formulation

Given a pre-trained NeRF, we aim at rendering it in real time on a variety of resource-constrained devices. Our goal is to transform said NeRF into a representation that runs on standard mesh-rasterization pipelines. A NeRF is a function  $R$  (parameterized by an MLP), which maps from a 3D location  $\mathbf{p}$  and a 2D viewing direction  $\mathbf{d}$  to an RGB emitted color  $\mathbf{c}$  and volume density  $\sigma$ . Formally, a NeRF implements  $R : (\mathbf{p}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma)$ . To render a pixel, NeRF follows the classical volume rendering formulation. In this formulation, the color assigned to a pixel with associated camera ray  $\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}$  is given by:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \sigma(\mathbf{r}(t)) \mathbf{c}(\mathbf{r}(t), \mathbf{d}) dt, \quad (1)$$

where  $T(t) = \exp\left(-\int_{t_n}^t \sigma(\mathbf{r}(s)) ds\right)$  represents the transmittance accumulated along  $\mathbf{r}$ .

To reach our goal of real-time rendering of NeRFs in resource-constrained devices, we must address the following three limitations of NeRFs: *(i)* their implicit nature conflicts with the explicit representations demanded by mesh-rasterization pipelines, which request a mesh  $\chi$  and a texture map  $\mathcal{M}$ , *(ii)* they compute color via Eq. (1), thus requiring expensive numerical estimation of the integral, and *(iii)* computing the individual integrands for this estimate requires evaluating the MLP that parameterizes  $R$ .

We seek a method that transforms  $R$ ’s knowledge into an alternative representation that circumvents these three limitations of NeRFs. That is, we require the method to output a rasterization-friendly representation, to compute pixel color with a single query, and to entirely dispose of MLPs for modeling view-dependent effects. Moreover, we require the method to effectively work on a wide variety of NeRFs.

### 4. Method: Re-ReND

We now introduce Re-ReND, a method enabling real-time rendering of a pre-trained NeRF on resource-constrained devices. Re-ReND receives a pre-trained NeRF  $R$  as input and transfers its knowledge to an alternative representation  $\mathcal{S}$  that is fast and inexpensive to render. The representation generated by Re-ReND achieves fast rendering by *(i)* representing the learnt scene as a mesh whose texture maps store embeddings, *(ii)* obtaining ray color via light fields (instead of radiance fields required by expensive volume rendering), and *(iii)* factorizing the light field computation as an MLP-free matrix multiplication. Please refer to Figure 2 for an overview of Re-ReND.

In essence, Re-ReND transforms a pre-trained NeRF  $R$  into a rasterization-friendly representation  $\mathcal{S} = \{\chi, \mathcal{M}\}$ . This representation is composed of a mesh  $\chi$ , and a set of four texture-map-like matrices  $\mathcal{M} = \{\mathcal{M}_u, \mathcal{M}_v, \mathcal{M}_w, \mathcal{M}_\beta\}$ . Each matrix  $\mathcal{M}_i$  stores representations we dub “light field embeddings”, where position-

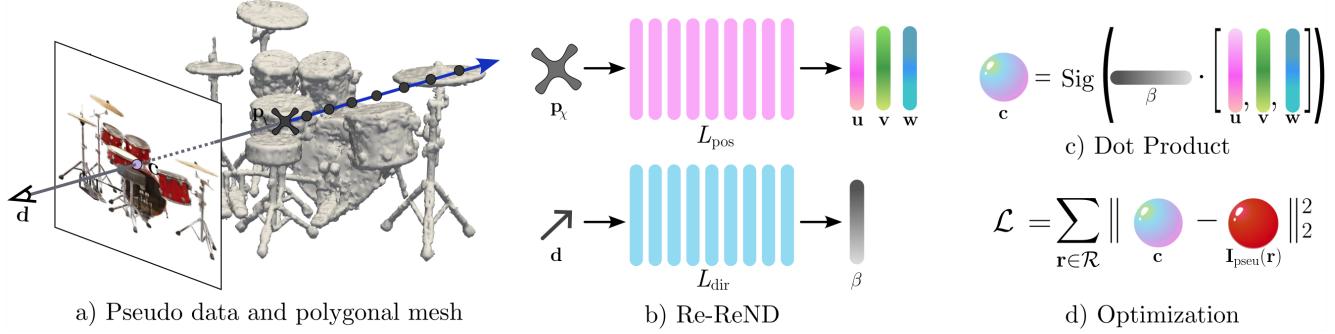


Figure 3. **Training Re-ReND.** **a)** Starting with a pre-trained NeRF, we extract a polygonal mesh and generate pseudo-images to train our factorized NeLF. These images are generated by rendering the NeRF from various points of view. **b)** Our factorized NeLF consists of two MLPs ( $L_{\text{pos}}$  and  $L_{\text{dir}}$ ) that *independently* compute position and direction embeddings.  $L_{\text{pos}}$  outputs position-dependent deep radiance embeddings ( $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$ ) for points *on* the mesh’s surface, while  $L_{\text{dir}}$  outputs direction-dependent embeddings ( $\beta$ ). Since this representation is amenable to “baking”, *i.e.* pre-computing and storing outputs, it allows us to dispose of MLPs for deployment. **c)** Under this formulation, computing pixel color amounts to a dot product of ( $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$ ) and  $\beta$ . **d)** We optimize our framework with an MSE reconstruction loss w.r.t. to the pseudo-images we extracted from the pre-trained NeRF.

dependent effects are modeled by the first three matrices, while the fourth one accounts for view-dependent effects. While obtaining the mesh  $\chi$  is straightforward from the densities generated by  $R$ , constructing  $\mathcal{M}$  is challenging.

In the next two sections, we describe how we obtain  $\mathcal{M}$ . In particular, Section 4.1 describes how we approximate  $R$  with a Factorized Neural Light Field (NeLF), and Section 4.2 explains how we extract  $\mathcal{M}$  from this NeLF.

#### 4.1. Representing $R$ with Factorized NeLFs

We construct  $\mathcal{M}$  by using the radiance field  $R$  as supervision to train a *Factorized* Neural Light Field (NeLF). Factorized NeLFs are light fields whose computation can be expressed as inexpensive matrix multiplications. We first explain the formulation for constructing a NeLF, and then the formulation for their factorization.

**NeLF.** We extract the color information learned by the NeRF  $R$  into a NeLF  $L$ . Light fields are the integral of radiance fields: while radiance fields reconstruct the individual integrands of Eq. (1) along the ray, light fields directly estimate the integral’s value, *i.e.* the ray’s color. Thus, light fields are more suitable for our purposes, as they directly yield  $C(\mathbf{r})$  instead of computing the integral of Eq. (1).

Formally, given a position  $\mathbf{p}$  and direction  $\mathbf{d}$ , the light field models the integrated radiance along the ray with origin at  $\mathbf{p}$  and direction  $\mathbf{d}$ , *i.e.*  $C(\mathbf{p} + t \mathbf{d})$ ,  $t \in [0, +\infty)$ . If the scene’s geometry is known, empty space can be skipped by considering a ray whose origin is the collision point of the ray and the scene’s geometry. We use our mesh  $\chi$  as an estimate to this geometry, and define  $\mathbf{p}_\chi^\mathbf{r} := \mathbf{r} \wedge \chi$  as the first intersection point between ray  $\mathbf{r}$  and mesh  $\chi$ . The NeLF  $L$  thus must approximate  $C$ , that is,

$$L(\mathbf{p}_\chi^\mathbf{r}, \mathbf{d}) \approx C(\mathbf{p} + t \mathbf{d}), \quad t \in [0, +\infty). \quad (2)$$

Inspired by [32], and using the formulation of Eq. (2), we can distill a NeRF  $R$  into a NeLF  $L$ . Specifically, we can

use  $R$  to compute the right hand side of this equation, and use these values as supervision to learn  $L$ . Thus, the NeLF is tasked with predicting the color of ray  $\mathbf{r}$  when evaluated at the collision point  $\mathbf{p}_\chi^\mathbf{r}$ . In practice, we sample camera views (and their associated set of rays  $\mathcal{R}$ ) and use  $R$  to compute a set of pseudo-images  $\mathcal{I}_{\text{pseu}}$ . Furthermore, we use  $\chi$  to pre-compute the collision points between the rays in  $\mathcal{R}$  and the scene. We then evaluate  $L$  at these points, and task it with predicting the colors given by  $\mathcal{I}_{\text{pseu}}$ .

**Factorized NeLFs.** For a given camera ray, the NeLF  $L$  predicts ray color when evaluated at the collision point of the camera ray and the scene. Because of the light field formulation, this approach can compute pixel colors with a single MLP query. However,  $L$ ’s evaluations are still expensive due to its MLP parameterization. Hence, we propose to rather learn a Factorized NeLF. The architecture of a Factorized NeLF is amenable to “baking”, *i.e.* pre-computing and storing network outputs, to dispose of MLPs at rendering time. We next describe Factorized NeLFs in detail.

Recall that our NeLF is a function  $L : (\mathbf{p}_\chi^\mathbf{r}, \mathbf{d}) \mapsto \mathbf{c}$ , parameterized by an MLP. This function maps a point (on the mesh)  $\mathbf{p}_\chi^\mathbf{r} \in \mathbb{R}^3$  and a ray direction  $\mathbf{d} \in \mathbb{R}^3$  to an RGB color  $\mathbf{c} \in \mathbb{R}^3$ . A Factorized NeLF  $L_F$  shares the same signature with  $L$ , but internally processes  $\mathbf{p}_\chi^\mathbf{r}$  and  $\mathbf{d}$  with two independent MLPs whose outputs produce color via an inexpensive matrix multiplication. We define these underlying direction- and position-dependent MLPs [11] as

$$\begin{aligned} L_{\text{pos}} : \mathbf{p}_\chi^\mathbf{r} &\mapsto [\mathbf{u}, \mathbf{v}, \mathbf{w}] \in \mathbb{R}^{D \times 3}, \\ L_{\text{dir}} : \mathbf{d} &\mapsto \beta \in \mathbb{R}^D. \end{aligned} \quad (3)$$

We refer to the vectors  $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$ , and  $\beta$  as *light field embeddings*. Our Factorized NeLF  $L_F$  is thus defined as

$$\begin{aligned} L_F(\mathbf{p}_\chi^\mathbf{r}, \mathbf{d}) &= \text{Sig}((L_{\text{pos}}(\mathbf{p}_\chi^\mathbf{r})^\top L_{\text{dir}}(\mathbf{d}))) \\ &= \text{Sig}([\mathbf{u}, \mathbf{v}, \mathbf{w}]^\top \beta), \end{aligned} \quad (4)$$

where  $\text{Sig}$  is the sigmoid function. Note, in Eq. (4), the position-dependent embeddings ( $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$ ) are weighed by the direction-dependent embeddings  $\beta$ . This formulation enables accelerated rendering via baking:  $L_{\text{pos}}$  and  $L_{\text{dir}}$  outputs are pre-computed and stored, so that  $L_F$  can be approximated at test time with an inexpensive MLP-free operation.

**Training Factorized NeLFs.** We train  $L_F$  to predict the colors given by the set of pseudo-images  $\mathcal{I}_{\text{pseu}}$  when evaluated at the point of collision with the scene. Formally, we train  $L_F$  to minimize the photo-metric loss

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \|L_F(\mathbf{p}_\chi^{\mathbf{r}}, \mathbf{d}) - \mathcal{I}_{\text{pseu}}(\mathbf{r})\|_2^2, \quad (5)$$

where  $\mathcal{R}$  is the set of rays with which the set of pseudo-images  $\mathcal{I}_{\text{pseu}}$  was computed, and  $\mathcal{I}_{\text{pseu}}(\mathbf{r})$  further denotes the color assigned to ray  $\mathbf{r}$  in the pseudo-images.

#### 4.2. Factorized NeLFs to Real-time Rendering

We now use our Factorized NeLF to build  $\mathcal{M}$ , the missing piece in our a rasterization-friendly representation  $\mathcal{S} = \{\chi, \mathcal{M}\}$ . Recall  $\mathcal{M} = \{\mathbf{M}_u, \mathbf{M}_v, \mathbf{M}_w, \mathbf{M}_\beta\}$  is a set of four matrices, where the first three matrices model position-dependent effects, and the fourth one view-dependent effects. Here we describe how we leverage our Factorized NeLF to construct  $\mathcal{M}$ , and then how we integrate this representation with mesh-rasterization pipelines.

**Constructing  $\mathcal{M}$ .** The Factorized NeLF formulation from Eq. (4) is amenable to baking, *i.e.* pre-computation and storage. Namely, if the outputs of  $L_{\text{pos}}$  and  $L_{\text{dir}}$  are baked, computing a ray’s color is reduced to an inexpensive MLP-free operation. To enjoy this MLP-free property when rendering with a mesh-rasterization pipeline, we use  $\mathcal{M}$  to store the baked outputs. Specifically, we pre-compute light field embeddings at various inputs and store them in the corresponding matrices  $\mathbf{M}_u$ ,  $\mathbf{M}_v$ ,  $\mathbf{M}_w$  and  $\mathbf{M}_\beta$ . For the position-dependent embeddings, we traverse  $\chi$ ’s faces, extract  $N_{\text{pos}}$  points in each face’s normalized  $uv$  coordinates, and then evaluate  $L_{\text{pos}}$  at such positions. Analogously, for the direction-dependent embeddings, we evaluate  $L_{\text{dir}}$  at independently sampled  $N_{\text{dir}}$  azimuth and elevation angles.

Once constructed  $\mathcal{M}$ , we are ready to integrate our representation  $\mathcal{S}$  with the rasterization pipeline.

**Integration with mesh rasterization.** We render our representation  $\mathcal{S} = \{\chi, \mathcal{M}\}$  on a standard graphics pipeline, enabling real-time rendering across devices, from mobiles all the way to VR headsets. Note that our scene representation is compatible with standard graphics pipelines: the required mesh is compatible with the  $\chi$  mesh extracted from  $R$ , and the texture are our four matrices in  $\mathcal{M}$ . In particular, as common in production rendering [4], we store the position-dependent info ( $\mathbf{M}_u$ ,  $\mathbf{M}_v$  and  $\mathbf{M}_w$ ) into a texture map-like with per-face textures, and the direction-dependent info

Device	Type	OS	GPU	Browser
Samsung S21	Phone	Android 13	Mali G78	Chrome
Motorola G9	Phone	Android 11	Adreno 610	Chrome
Galaxy S6	Tablet	Android 13	Mali G72 MP3	Firefox
Dell	Laptop	Windows 10	Integrated GPU	Firefox
Gaming Lap.	Laptop	Windows 10	NVIDIA GF RTX 2060	Firefox
Desktop	Desktop	Ubuntu 18.04	NVIDIA GF RTX 3090	Chrome
Meta Quest P.	Headset	Oculus	Adreno 650	–

**Table 1. Testing Devices.** We compare Re-ReND against other methods on a set of representative devices with a wide range of compute capabilities. Our devices include low- to high-end mobile phones, tablets, laptops, desktop computers, and a VR headset.

( $\mathbf{M}_\beta$ ) in a texture map sorted by its corresponding sampling angles. We then quantize all baked outputs, and export them as PNG files. Finally, we deploy our rendering pipeline within a fragment shader for compatibility with standard rasterization frameworks (please refer to Figure 3).

## 5. Experiments

Next, we conduct an extensive study of Re-ReND performance on multiple scenes and devices.

**Datasets.** We experiment on both synthetic and real data by using two standard datasets:

- **Synthetic 360° dataset** [3], eight synthetic scenes with intricate geometries and non-Lambertian materials. Each scene has 100 views for training and 200 views for testing, both at a resolution of 800px  $\times$  800px.
- **Tanks and Temples (T&T) dataset** [15], an unbounded dataset consisting of hand-held 360° captures of four large-scale scenes. We use the dataset configuration of [37].

**Devices.** Our main goal is to test our method on hardware-constrained devices such as mobile phones, tablets, and VR headsets. Nonetheless, for completeness, we also test Re-ReND on more powerful laptops and desktops. In total, we test Re-ReND on seven devices reported in Table 1.

### 5.1. Implementation details

**Pre-trained NeRFs.** For the pre-trained NeRF models  $R$ , we use MipNeRF [2] (in the Synthetic 360° dataset), and NeRF++ [37] (in the T&T dataset).

**Meshing.** Obtaining the mesh  $\chi$  from a pre-trained NeRF requires distilling the learnt geometry. For the Synthetic 360° dataset, we run Marching Cubes [19] on a density grid of side  $K = 256$ , except for the *ficus* object, in which we use  $K = 512$  to capture finer geometric details. For the T&T dataset, we use a grid with  $K = 512$ . We remove small connected components resulting from noisy estimates, and decimate the meshes to around 400k faces. Finally, we enclose unbounded scenes within a dome.

**Pseudo-Image Generation.** Training the Factorized NeLF via Eq. (5) requires the set  $\mathcal{I}_{\text{pseu}}$  of pseudo-images. We obtain these images by using the NeRF to render 10k images from random camera poses for each scene.

Device	Method	Synthetic 360°					Unbounded 360°				
		FPS	GPU (MB)	Disk (MB)	PSNR (dB)	Mesh (F/V)	FPS	GPU (MB)	Disk (MB)	PSNR (dB)	Mesh (F/V)
Samsung	M-NeRF [6]	41.7	570	125.8	<b>30.9</b>	494k/224k	22.8	943	311.3	15.6	494k/224k
	SNeRG [13]	†	3,627	<b>87.0</b>	30.4	N/A	†	4,386	324.8	14.0	N/A
	Ours	<b>54.7</b>	<b>532</b>	199.0	29.0	<b>205k/99k</b>	<b>33.5</b>	<b>816</b>	<b>288.3</b>	<b>17.9</b>	<b>244k/117k</b>
Desktop	M-NeRF [6]	124.3	7,350	125.7	<b>30.9</b>	494k/224k	14.3	7,970	311.3	15.6	332k/543k
	SNeRG [13]	49.1	3,963	<b>87.0</b>	30.4	N/A	9.9	5,102	324.8	14.0	N/A
	Ours	<b>701.4</b>	<b>1,501</b>	199.0	29.0	<b>205k/99k</b>	<b>235.3</b>	<b>1,800</b>	<b>288.3</b>	17.9	<b>244k/117k</b>
	Ours (Large)	329.6	2,964	800.7	30.1	<b>205k/99k</b>	118.6	3,445	1,142.9	<b>18.0</b>	<b>244k/117k</b>
	NeRF [20]	7e-4	OOM	13.3	31.0	N/A	7e-4	OOM	13.3	19.0	N/A
	Mip-NeRF [2]	1e-2	OOM	7.4	33.1	N/A	—	—	—	—	—
	NeRF++ [37]	—	—	—	—	—	8e-4	OOM	29.0	20.1	N/A

Table 2. **Quantitative Results.** We report rendering speed, GPU memory a disk requirements, PSNR, and number of mesh faces and vertices on a mobile device and a desktop computer for both datasets. The last three rows include baseline methods, which do not run on mobile devices. Rendering speed is measured at a resolution of 8K on a desktop and 800px on a mobile device (Samsung). GPU memory requirements are calculated for rendering a full frame. Methods unable to render an entire frame are denoted as Out of Memory (OOM). Along with our base model, we report metrics on a variation “Ours (Large)” with higher texel count (10) and larger embedding dimension (64). We achieve a major speedup along with a higher quality rendering on the Unbounded 360 dataset and a slight degradation of quality in the Synthetic 360 dataset. “†” denotes the method was unable to run on device while “—” denotes a missing implementation.

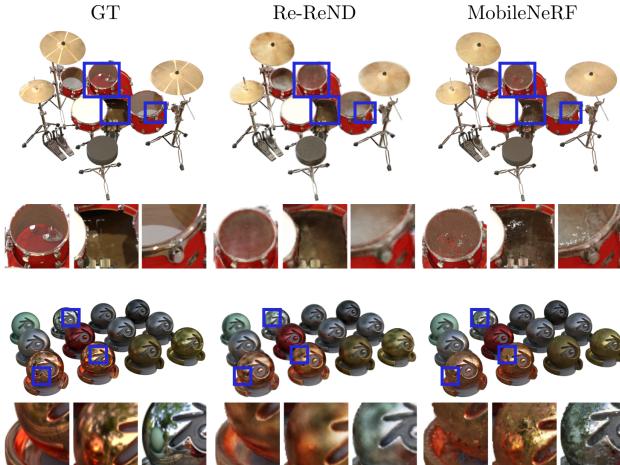


Figure 4. **Qualitative Results on Synthetic Scenes.** We show renderings of two synthetic scenes to compare Re-ReND and MobileNeRF [6]. In both scenes, we can see how Re-ReND accurately more reproduces light effects like shininess and reflections. Notice that MobileNeRF fails to model such scene producing holes and high frequency noise.

**Factorized NeLF Training.** We implement the position- and direction-dependent MLPs ( $L_{\text{pos}}$  and  $L_{\text{dir}}$ ) by following [32], and thus employ intensive residual blocks [12] and deep architectures of 88 layers. We train these MLPs with hard-ray sampling [32] and learning rate warm-up strategies. We train with a batch size of 200k rays for 2.5 days on one NVIDIA A100 GPU.

**Baking Light Field Embeddings.** For baking  $\beta$ , we uniformly sample elevation and azimuth angles in  $[0-180^\circ]$  and  $[0-360^\circ]$ , respectively. We use 1024 samples for synthetic scenes and 2048 for real scenes. When quantizing  $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$ , and  $\beta$  for their storage as PNG files, we perform per-channel min-max normalization. Unless otherwise stated, we use  $18 = \lceil 6 \times 6/2 \rceil$  texels per triangle face.

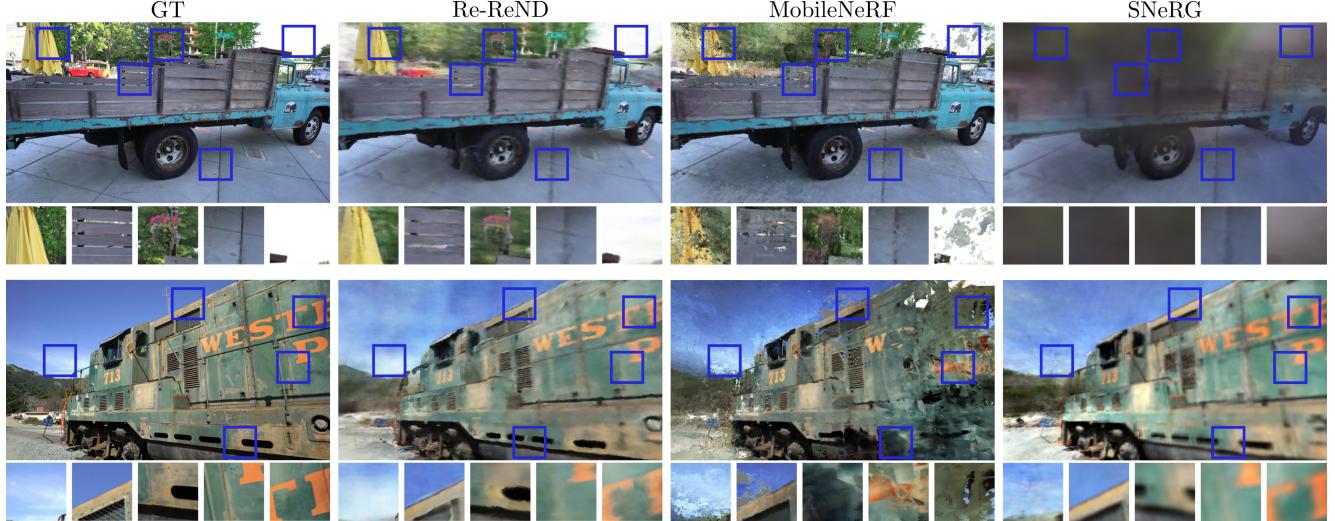
**Rendering with Shaders.** Since Re-ReND requires no MLP queries, we implement it in a simple fragment shader.

This implementation allows deploying Re-ReND across not only various devices, but also different graphics frameworks. Notably, this implementation allows us to deploy on a VR headset (Meta Quest Pro), which runs Unity shaders. The shader computes color by combining the position- and direction-dependent embeddings according to Eq. (4), where each embedding is queried from its corresponding texture map. In turn, each texture map is a  $2 \times 4$  grid stored in a 4-channeled PNG image, thus fully accounting for our default embedding dimension  $D = 32 = 2 \times 4 \times 4$ . We obtain the position embeddings by indexing  $\mathbf{M}_{\mathbf{u}, \mathbf{v}, \mathbf{w}}$  with the fragment’s  $uv$  coordinates, while the direction embeddings are obtained by indexing  $\mathbf{M}_\beta$  with the azimuth and elevation angles. Since texture values are 8-bit quantized, we map to the original range by reverting the per-channel min-max normalization. We provide a full implementation of our shaders in the [Appendix](#).

## 5.2. Main Results

We compare against two methods intended for fast rendering, SNeRG [13] and MobileNeRF [6].

**Quantitative Results.** The first three rows of Table 2 show how Re-ReND can achieve real-time rendering when deployed on a mobile. For both datasets, we find that, while SNeRG achieves reasonable disk usage and photo-metric quality, it is ultimately incapable of interactive rendering. On the Synthetic 360° dataset, we find that Re-ReND can render at over 54 FPS, outperforming Mobile-NeRF by more than 30%. Furthermore, such fast rendering speed comes at a negligible drop in performance of  $\sim 1$  dB. Moreover, these benefits over MobileNeRF are accompanied by simpler meshes (*i.e.* faces and vertex quantity). On the realistic Unbounded 360° dataset, our method outperforms competing approaches by even larger margins than on synthetic data. In particular, we improve upon MobileNeRF *both* in rendering quality and speed. That is,



**Figure 5. Qualitative Results of Real Scenes.** We show renderings of challenging real scenes using Re-ReND, MobileNeRF [6], and SNeRG [13]. Our framework can generate sharper and more accurate renders. In contrast, both MobileNeRF and SNeRG struggle with foreground and background artifacts, and fail to reconstruct fine details such as the writings on the train (bottom images).

while SNeRG and MobileNeRF achieve a PSNR of 14.0 and 15.6, respectively, Re-ReND attains 17.9. Impressively, this outstanding gain in photo-metric quality of over 2 dB is accompanied by an also impressive superiority in rendering speed of over 10 FPS over MobileNeRF (22.79 vs. 33.46 FPS). The gains we achieve can be attributed to our light field-based formulation, that allows us to work with smaller meshes while still maintaining high directional expressivity. The bottom rows of Table 2 report the performance of all methods in a non-constrained scenario, which is a desktop. When provided with such larger computational resources, our method’s speed improves both in absolute and relative terms. Specifically, while the gap in mobiles w.r.t. MobileNeRF was  $\sim 30\%$  (on the Synthetic dataset), this gap jumps to over 460% on the desktop. This increased gap is even more pronounced in the Unbounded dataset: the gap in mobiles was  $\sim 45\%$  (from 22 FPS to 33), while the gap in desktop becomes over 1,500%. The larger computational resources also allow us to test a larger version of Re-ReND, which achieves remarkable PSNRs of 30.1 and 18.0 in the Synthetic and Unbounded datasets, respectively, while also remaining much faster than MobileNeRF. The last three rows of Table 2 report foundational methods (*i.e.* NeRF, Mip-NeRF and NeRF++) as reference of high quality, although these methods are inefficient for rendering.

**Qualitative Results.** We next compare Re-ReND performance against the other real-time rendering methods. We show results for both synthetic (Figure 4) and real scenes (Figure 5). Note Re-ReND higher quality, particularly in real scenes, where our method preserves sharper object boundaries, and crisper surface details. Re-ReND allows real-time rendering while preserving strong image quality.

Device	Synthetic 360°			Unbounded 360°		
	SNeRG	M-NeRF	Re-ReND	SNeRG	M-NeRF	Re-ReND
Samsung S21	†	41.7	<b>54.7</b>	†	22.8	<b>33.5</b>
G9	†	9.7	<b>10.4</b>	†	3.5	<b>7.7</b>
Galaxy S6	N/A	18.1	<b>26.6</b>	†	6.2	<b>20.4</b>
Dell	†	49.8	<b>75.3</b>	†	16.9	<b>54.0</b>
Gaming	197.9	496.0	<b>697.3</b>	46.0	169.9	<b>516.6</b>
Desktop	502.1	762.3	<b>1,013.2</b>	141.1	389.8	<b>925.4</b>
Headset	†	—	<b>74.0*</b>	†	—	<b>74.0*</b>

**Table 3. On-device rendering speed (FPS).** We compare the rendering speed of Re-ReND against MobileNeRF (M-NeRF) [6] and SNeRG [13] across devices. Conventions: “\*” means the device’s FPS limit was reached, “—” means missing implementation, and “†” means the method failed to run. For all devices and datasets, Re-ReND provides the fastest rendering speeds, usually by a large margin. Furthermore, we highlight that Re-ReND is capable of real-time rendering even on VR headsets.

### 5.3. Rendering Speed on Devices

We measure frames-per-second (FPS) in seven devices (via the same procedure as [6]), and report results in Table 3. Our results illustrate how Re-ReND significantly outperforms other methods in rendering speed. The advantages in performance are particularly large in unbounded scenes. In this scenario, Re-ReND is, on average  $2.6\times$  faster than MobileNeRF. For synthetic scenes, our method provides sizable speed gains of 35%. Importantly, we find that Re-ReND is capable of real-time rendering in a VR headset even reaching the device’s limit of 74 FPS.

### 5.4. Quality vs. Representation Size

Two main factors affect the size of our representation: the texel count and the dimensionality of the light field embeddings ( $D$  in Eq. (3)). Here we study how these factors, in turn, affect the photo-metric quality of Re-ReND. **Texel count.** We examine the effect that varying the num-

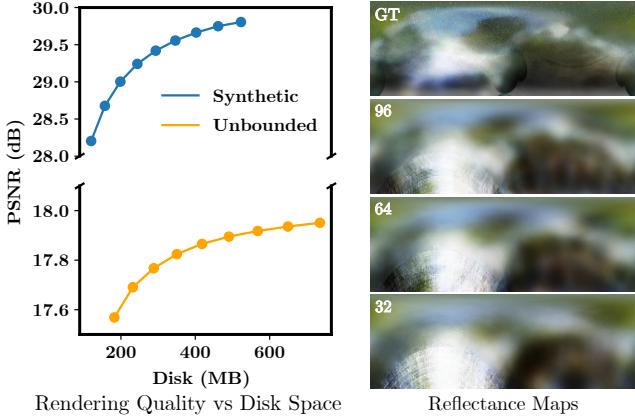


Figure 6. **Left:** We compare how PSNR varies as the number of texels per face increases for both synthetic and unbounded scenes. It is possible to trade-off disk space for higher quality renderings by increasing the number of texels. **Right:** We visualize color as a function of elevation (y-axis) and azimuth (x-axis) for a surface point  $x, y, z$  on the Materials scene from the Synthetic dataset, for different embedding dimensions.

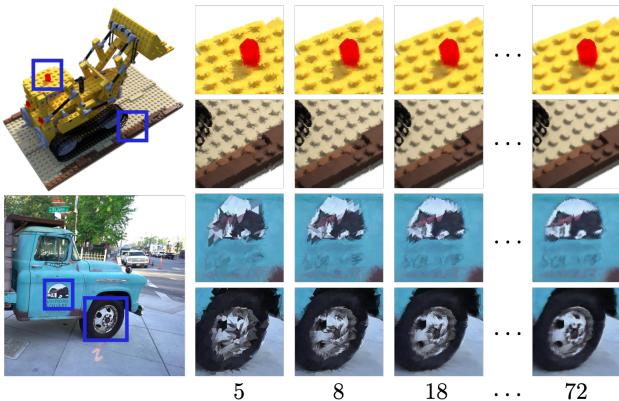


Figure 7. **Qualitative Effect of Varying Number of Texels.** We show renderings of synthetic and real scenes with an increasing number of texels per triangle face. Higher texels per face result in crispier renderings, although it is possible to appreciate directional effects such as specular highlights even with low texel counts.

ber of texels assigned to each face in the mesh has on rendering quality and disk space. Figure 6 (left) and Figure 7 show that, for both datasets, an increased texel count is accompanied by a drastic rise in quality of the reconstruction and disk space. Embedding dimensionality. We now examine the effect that varying the dimensionality of the light field embeddings has on the ability to represent light effects. Reflectance maps in Figure 6 (right) demonstrate that Re-ReND can model challenging view-dependent effects even with low-dimensional embeddings, but larger embeddings enable more complex reflections. Together, our experiments demonstrate that exchanging disk space for renderings of higher quality, and suggest that increasing the texel count can improve rendering quality.

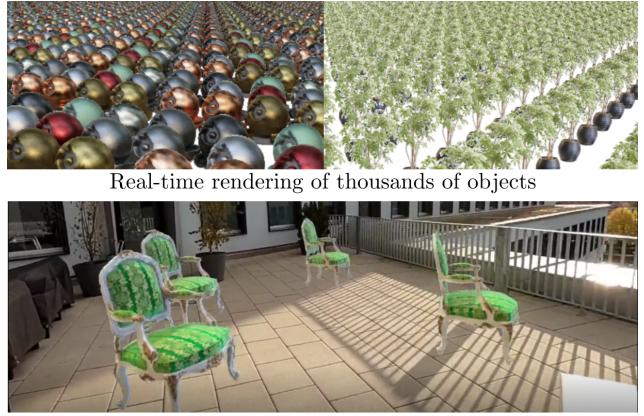


Figure 8. **Scene Composition.** **Top:** The outcome of rendering 2500 materials and ficus objects together in single scenes. **Bottom:** An AR application showcase of Re-ReND with four chairs in a real-world setting using an AR/VR headset. See the [supplementary video](#) for more details.

## 5.5. Compositional scenes

In Figure 8, we showcase the practical application of Re-ReND for scene composition. Our approach enables efficient rendering of 2500 materials and ficus objects in single scenes at 130 FPS each on a desktop. Additionally, we demonstrate an AR application that uses an AR/VR headset to render four chairs in real-time in a real-world setting. This exemplifies our approach’s ability to seamlessly blend virtual and real environments, unlocking new possibilities for immersive visual experiences.

**Additional Results.** Finally, we report detailed qualitative and quantitative results, validation of view-dependent effects, sensitivity to geometry variations and photo-metric quality depending on the dimensionality  $D$  of Re-ReND in the [Appendix](#).

## 6. Conclusions and Limitations

We present Re-ReND, a method enabling Real-time Rendering of NeRFs across Devices. Our method receives a pre-trained NeRF as input and transforms it into a representation that renders real-time in various devices, from mobile phones to VR headsets. Our approach achieves fast rendering by leveraging standard graphics pipelines, obtaining pixel color with a single query, and avoiding MLP evaluations altogether. We achieve these objectives by distilling NeRF and extracting density into a mesh, and color information into matrices for efficient light field computation.

We test Re-ReND on a variety of resource-constrained devices, and find outstanding trade-off between rendering time and photo-metric quality. In particular, Re-ReND achieves over  $2.6\times$  FPS improvements when rendering challenging and unbounded real scenes. Furthermore, our method allows rendering of these scenes on VR headsets in

real-time, a task out of hand for current NeRF architectures.

Limitations of our approach include relying on pre-trained NeRF for reasonable scene-geometry reconstructions and the requirement for large embeddings to achieve sensible reconstructions. Improvements in these areas could further enhance rendering speed and quality.

**Acknowledgments.** This work was supported by the King Abdullah University of Science and Technology (KAUST) Office of Sponsored Research through the Visual Computing Center (VCC) funding, as well as, the SDAIA-KAUST Center of Excellence in Data Science and Artificial Intelligence (SDAIA-KAUST AI).

## References

- [1] Benjamin Attal, Jia-Bin Huang, Michael Zollhöfer, Johannes Kopf, and Changil Kim. Learning neural light fields with ray-space embedding networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [2] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields, 2021.
- [3] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.
- [4] Brent Burley and Dylan Lacewell. Ptex: Per-face texture mapping for production rendering. In *Computer Graphics Forum*, 2008.
- [5] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022.
- [6] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022.
- [7] Forrester Cole, Kyle Genova, Avneesh Sud, Daniel Vlasic, and Zhoutong Zhang. Differentiable surface rendering via non-differentiable sampling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.
- [8] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.
- [9] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *ACM Siggraph Computer Graphics*, 22(4):65–74, 1988.
- [10] Brandon Yushan Feng and Amitabh Varshney. Signet: Efficient neural representation for light fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.
- [11] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [13] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021.
- [14] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.
- [15] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.
- [16] Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1440–1449, 2021.
- [17] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996.
- [18] David B Lindell, Julien NP 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*, pages 14556–14565, 2021.
- [19] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.
- [20] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.
- [21] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.
- [22] T. Neff, P. Stadlbauer, M. Parger, A. Kurz, J. H. Mueller, C. R. A. Chaitanya, A. Kaplanyan, and M. Steinberger. Donerf: Towards real-time rendering of compact neural radiance fields using depth oracle networks. *Computer Graphics Forum*, 2021.
- [23] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [24] 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 *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [25] 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*, pages 14153–14161, 2021.
- [26] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.
- [27] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022.
- [28] Vincent Sitzmann, Semon Rezhnikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *Advances in Neural Information Processing Systems*, 2021.
- [29] Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. Light field neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.
- [30] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022.
- [31] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020.
- [32] Huan Wang, Jian Ren, Zeng Huang, Kyle Olszewski, Menglei Chai, Yun Fu, and Sergey Tulyakov. R2l: Distilling neural radiance field to neural light field for efficient novel view synthesis. *arXiv preprint arXiv:2203.17261*, 2022.
- [33] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [34] Suttisak Wizadwongsu, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwanjanakorn. Nex: Real-time view synthesis with neural basis expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [35] 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 *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000.
- [36] 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.
- [37] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020.
- [38] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–595, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.
- [39] 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 Transactions on Graphics (TOG)*, 2021.