Neural Rays for Occlusion-aware Image-based Rendering

Yuan Liu[1] Sida Peng[2] Lingjie Liu[3] Qianqian Wang[4] Peng Wang[1]
Christian Theobalt[3] Xiaowei Zhou[2] Wenping Wang[5]
[1]The University of Hong Kong [2]Zhejiang University [3]Max Planck Institute for Informatics
[4]Cornell University [5]Texas A&M University

Abstract

We present a new neural representation, called Neural Ray *(NeuRay), for the novel view synthesis task. Recent works construct radiance fields from image features of input views to render novel view images, which enables the generalization to new scenes. However, due to occlusions, a 3D point may be invisible to some input views. On such a 3D point, these generalization methods will include inconsistent image features from invisible views, which interfere with the radiance field construction. To solve this problem, we predict the visibility of 3D points to input views within our NeuRay representation. This visibility enables the radiance field construction to focus on visible image features, which significantly improves its rendering quality. Meanwhile, a novel consistency loss is proposed to refine the visibility in NeuRay when finetuning on a specific scene. Experiments demonstrate that our approach achieves state-of-the-art performance on the novel view synthesis task when generalizing to unseen scenes and outperforms per-scene optimization methods after finetuning. Project page: https://liuyuan-pal.github.io/NeuRay/*

1. Introduction

Novel View Synthesis (NVS) is an important problem in computer graphics and computer vision. Given a set of input images with known camera poses, the goal of NVS is to synthesize images of the scene from arbitrary virtual camera poses. Recently, neural rendering methods have achieved impressive improvements on the NVS problem compared to earlier image-based rendering methods [14, 19, 30]. Neural Radiance Field (NeRF) [31] shows that photo-realistic images of novel views can be synthesized by volume rendering on a 5D *radiance field* encoded in a neural network which maps a position and a direction to a density and a color. However, these methods cannot generalize to unseen scenes as they learn scene-specific networks, which usually take hours or days for a single scene.

Recent works [5, 55, 56, 63] propose NeRF-like neural

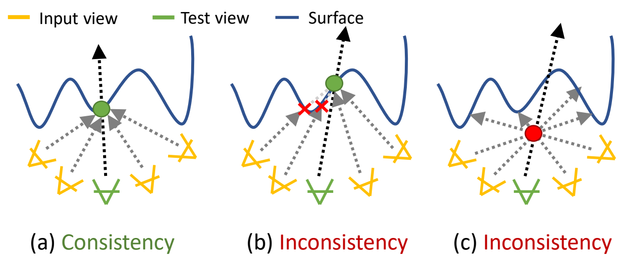

Figure 1. (a) Without occlusions, local image features are consistent on the surface point. (b) Local image features are inconsistent on a surface point due to occlusions. (c) Local image features are inconsistent on a non-surface point. Generalization methods will correctly assign a large density to the surface point in (a) due to the feature consistency. However, when features are not very consistent in (b) and (c), it is relatively hard for these methods to correctly determine the density.

rendering frameworks that can generalize to unseen scenes. Given a set of input views, they construct a radiance field on-the-fly by extracting local image features on these views and matching multi-view features to predict colors and density on 3D points. This is similar to traditional stereo matching methods [47, 60] that check the multi-view feature consistency to find a surface point, as shown in Fig. 1 (a). However, when features are not consistent on a point, it is relatively hard for these methods to correctly determine whether such inconsistency is caused by occlusions as shown in Fig. 1 (b) or this point is a non-surface point as shown in Fig. 1 (c), leading to rendering artifacts.

To address this problem, we introduce a new neural representation called Neural Rays (NeuRay) in this paper. NeuRay consists of pixel-aligned feature vectors on every input view. On a camera ray emitting from a pixel on the input view, the associated NeuRay feature vector on this pixel is able to predict visibility to determine whether a 3D point at a specific depth is visible or not. With such visibility, we can easily distinguish the occlusion-caused feature inconsistency from the non-surface-caused feature inconsistency in Fig. 1, which leads to more accurate radiance field construction and thus better rendering quality on difficult scenes with severe self-occlusions.

A key challenge is how to estimate the visibility in an unseen scene. This is a chicken-and-egg problem because the estimation of visibility requires knowing the surface locations while the estimated visibility is intended for better surface estimation in the radiance field construction. To break this cycle, we propose to apply well-engineered multi-view stereo (MVS) algorithms, like cost volume construction [60] or patch-matching [47], to reconstruct the scene geometry and then extract the pixel-aligned feature vectors of NeuRay from the reconstructed geometry. In the end, NeuRay will be used in the computation of the visibility to improve the radiance field construction.

Another problem is how to parameterize such visibility in NeuRay. A direct way is to predict the densities along the camera ray from the view to the 3D point and then accumulate these densities to compute the transmittance as the visibility like NeRF [31]. However, computing visibility with this strategy is computationally impractical because given N input views and a 3D point, we should accumulate the density along all N camera rays from every input view to this 3D point, which means we need to sample K points on every camera ray and evaluate the density on all $N \times K$ sample points. To reduce the computation complexity, we directly parameterize the visibility with a Cumulative Distribution Function (CDF) in NeuRay, which avoids density accumulation along rays and only requires N network forward passes to compute the visibility of all N input views.

NeuRay not only help the radiance field construction on unseen scenes but also be able to refine itself by finetuning on a specific scene with a novel consistency loss. Since both the NeuRay representation and the constructed radiance field depict the scene geometry, we propose a loss to enforce the consistency between the surface locations from the NeuRay representation and those from the constructed radiance field. This loss enables NeuRay to memorize the scene geometry predicted by the radiance field. At the same time, the memorized scene geometry in NeuRay will in turn improve the radiance field construction by providing better occlusion inference.

We conducted extensive experiments on the NeRF synthetic dataset [31], the DTU dataset [18] and the LLFF dataset [30] to demonstrate the effectiveness of NeuRay. The results show that 1) without scene-specific optimization, our method already produces satisfactory rendering results that outperforms other generalization methods by a large margin; 2) finetuning NeuRay produces much superior results than finetuning other generalization models and achieve even better rendering quality than NeRF [31]. Moreover, we can speed up rendering with the help of NeuRay by caching features on input views and predicting coarse surface locations, which costs $\sim$3 seconds to render an image of size 800×600.

2. Related works

2.1. Image-based rendering

Many works [3, 11, 14, 15, 19, 21, 22, 41] have focused on blending input images with geometry proxies to synthesize novel views. Conventional light field-based methods [7, 11] reconstruct a 4D plenoptic function from densely sampled views, which achieve photo-realistic rendering results but typically have a limited renderable range. To extend the renderable range, some works [3, 37] seek the help of 3D proxy geometry from multi-view stereo (MVS) methods [47]. With the development of deep learning techniques, some methods [6, 14, 19, 41, 42, 54, 59] introduce convolutional neural networks (CNNs) to replace hand-crafted components of the image-based rendering (IBR). One common challenge for the IBR methods is the sensitivity to the quality of estimated depth maps [6]. Our method also belongs to the category of image-based rendering and also uses cost volumes or estimated depth from MVS methods. However, our method can be trained from scratch without the help of external MVS algorithms and also can be finetuned on a scene to remedy reconstruction errors of MVS.

2.2. Neural scene representation

Recently, instead of estimating an external 3D proxy geometries, some methods have attempted to construct explicit trainable 3D representations from input images with differentiable renderers, such as voxels [28, 48], textured meshes [12, 25, 26, 53], and point clouds [1, 21, 43, 58]. Then, novel view can be synthesized from the constructed 3D representations. To further improve the rendering resolution, some methods [20, 23, 27, 31, 34, 36, 49] resort to pure neural fields encoded by neural networks to represent 3D scenes. Our method represents a scene by a ray-based representation for the NVS task. PIFu [44] and its follow-ups [13, 16, 45] also use ray-based representations to reconstruct human shapes. NeRF [31] renders photo-realistic images by volume rendering on a radiance field. Many following works [9, 23, 24, 29, 33, 35, 39, 50, 57, 62] have attempted to improve NeRF in various aspects. Among these, NeRV [50] also uses a visibility prediction for efficient relighting. In comparison, visibility in our method is used in efficiently constructing a radiance field on-the-fly.

Generalization volume rendering. NeRF usually takes a long time to train on each new scene. To address this, recent works [4, 5, 38, 40, 55, 56, 63] introduce generalizable rendering methods which construct a radiance field on-the-fly. Also, some works speed up NeRF-training by meta-learning [2, 52] or voxel [32, 51, 61]. Our method also bases on constructing radiance field on-the-fly to generalize to unseen scenes. The difference is that our method uses a ray-based representation for occlusion inference which greatly improves the rendering quality.

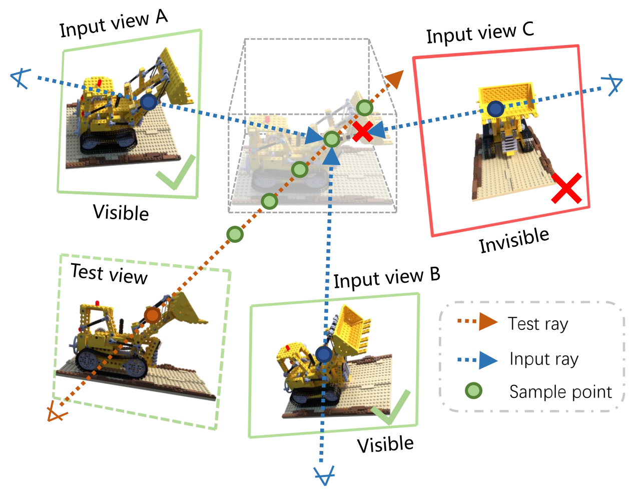

Figure 2. Constructing a radiance field on-the-fly from input views to synthesize the test view image by volume rendering. We first sample points on the test ray, then aggregate local features of input views to determine the alpha values and colors of sample points, and finally accumulating colors by volume rendering to compute the output color. Our method constructs NeuRay on input views to predict the visibility to sample points so that it allows occlusion-aware feature aggregation on these sample points.

3. Method

Given N input views of a scene with known camera poses, our goal is to render images on arbitrary novel test views. Before introducing NeuRay, we first review the volume rendering on a radiance field [31].

3.1. Volume rendering

In our method, images of test views are synthesized by volume rendering as shown in Fig. 2. Assuming a camera ray emitting from the test view, called a *test ray*, is parameterized by $\boldsymbol{p}(z) = \boldsymbol{o}+z\boldsymbol{r}$, $z \in \mathbb{R}^+$, where $\boldsymbol{o}$ is the start point at the camera center, and $\boldsymbol{r}$ is the unit direction vector of the ray. First, we sample K_t points $\{\boldsymbol{p}_i \equiv \boldsymbol{p}(z_i)|i=1,...,K_t\}$ with the increasing values z_i along the ray. Then, the color for the associated pixel of this camera ray is computed by

$$\boldsymbol{c} = \sum_{i=1}^{K_t} \boldsymbol{c}_i h_i, \tag{1}$$

where $\boldsymbol{c} \in \mathbb{R}^3$ is the rendered color for the pixel, $\boldsymbol{c}_i \in \mathbb{R}^3$ is the color of the sample point $\boldsymbol{p}_i$, and $h_i \in \mathbb{R}$ the hitting probability that the ray is not occluded by any depth up to the depth z_i and hits a surface in the range (z_i, z_{i+1}). Thus, the hitting probability h_i can be computed by

$$h_i = \prod_{k=1}^{i-1}(1-\alpha_k)\alpha_i, \tag{2}$$

where α_i is the alpha value in the depth range (z_i, z_{i+1}). In order to render a novel image using Eq. (1) and Eq. (2), we construct a radiance field to compute α_i and $\boldsymbol{c}_i$.

3.2. Occlusion-aware radiance field construction

Radiance field construction on-the-fly. In contrast to NeRF [31] which learns a scene-specific neural radiance field, generalization rendering methods [5, 55, 56, 63] construct a radiance field on-the-fly by aggregating local features. Given a 3D point $\boldsymbol{p}_i \in \mathbb{R}^3$ as shown in Fig. 2, these methods first extract features on input views by a CNN and then aggregate features of input views on this point by

$$\boldsymbol{f}_i = \mathcal{M}(\{\boldsymbol{f}_{i,j}|j=1,...,N\}), \tag{3}$$

where $\boldsymbol{f}_{i,j}$ is the local image feature of the sample point $\boldsymbol{p}_i$ projected on the j-th input view, $\mathcal{M}$ is a network which aggregates the features from different views to produce a feature $\boldsymbol{f}_i$. Then, the alpha value α_i and color $\boldsymbol{c}_i$ for this point $\boldsymbol{p}_i$ will be decoded from the aggregated feature $\boldsymbol{f}_i$ by other networks [5,55,56,63]. We provide more details about this in the supplementary material.

Occlusion-aware construction. The proposed method also constructs a radiance field on-the-fly as previous methods. Additionally, we predict a visibility term $v_{i,j}$ illustrating the j-th input view is visible or not to this 3D point $\boldsymbol{p}_i$ for the occlusion-aware feature aggregation

$$\boldsymbol{f}_i = \mathcal{M}(\{\boldsymbol{f}_{i,j}, v_{i,j}|j=1,...,N\}). \tag{4}$$

In this case, the aggregation network $\mathcal{M}$ is able to focus on visible views in the aggregated feature $\boldsymbol{f}_i$ and reduces the interference from invisible views. In the following, we will introduce our NeuRay for the computation of visibility $v_{i,j}$.

3.3. NeuRay representation

Given a camera ray emitting from a input view, called an *input ray*, NeuRay is able to predict the visibility function $v(z)$ indicating a point at depth z is visible or not for this input ray as shown in Fig. 3. On every input view, NeuRay is represented by a visibility feature map $\boldsymbol{G} \in \mathbb{R}^{H\times W\times C}$. Denoting $\boldsymbol{g} \in \mathbb{R}^C$ as a corresponding feature vector for the given input ray on $\boldsymbol{G}$, we will compute the visibility $v(z)$ of this input ray from $\boldsymbol{g}$. Obviously, a valid visibility function $v(z)$ should be non-increasing on z and $0 \le v(z) \le 1$. In the following, we discuss how to parameterize the visibility function $v(z)$ from $\boldsymbol{g}$. Then, we will introduce how to compute $\boldsymbol{G}$ in Sec. 3.4 and Sec. 3.5.

Visibility from occlusion probability. We represent the visibility function with a Cumulative Density Function (CDF) $t(z)$ by $v(z) = 1 - t(z)$ as shown in Fig. 3, which is parameterized as a mixture of logistics distributions

$$t(z;\{\mu_i,\sigma_i,w_i\}) = \sum_{i}^{N_l} w_i S((z-\mu_i)/\sigma_i), \tag{5}$$

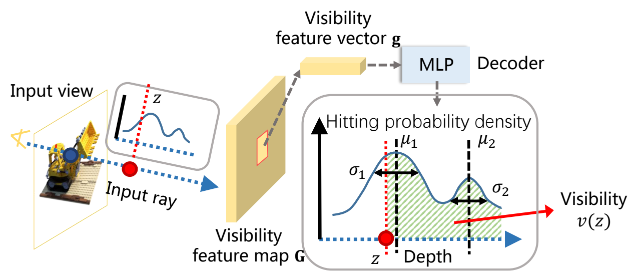

Figure 3. Visibility computation in NeuRay. NeuRay consists of a visibility feature map $\boldsymbol{G}$ on every input view. Every feature vector $\boldsymbol{g}$ on $\boldsymbol{G}$ can be decoded to a mixture of logistics distributions by an MLP. The distribution illustrates the visibility function $v(z)$ of the input ray emitting from the location of $\boldsymbol{g}$, which is the area under the curve after the depth z.

where we mix N_l logistics distributions, μ_i is the mean of i-th logistics distribution, σ_i the standard deviation, w_i the mixing weight with $\sum_i w_i = 1$, $S(\cdot)$ a sigmoid function. All parameters $[\mu_i, \sigma_i, w_i] = \mathcal{F}(\boldsymbol{g})$ are decoded from the feature $\boldsymbol{g}$ by an MLP $\mathcal{F}$. As a CDF, $t(z)$ is non-decreasing, so that $1 - t(z)$ forms a valid visibility function.

$t(z)$ actually corresponds to an *occlusion probability* that the input ray is occluded before a depth z and we call the corresponding probability density function (PDF) *hitting probability density*. The visibility $v(z)$ is actually the area under the PDF curve after z. In general, a ray will only hit one surface so one logistics distribution will be enough. However, using a mixture of logistics distributions improves performance when the ray hits on semi-transparent surfaces or edges of surfaces.

Discussion. Alternatively, we may parameterize visibility with a NeRF [31]-like density. However, computing visibility with this strategy is too computationally intensive. In this formulation, we directly decode a density $d(z) = \phi(z; \boldsymbol{g})$ from $\boldsymbol{g}$ using a MLP ϕ. To compute the visibility $v(z)$, we need to first sample K_r depth $\{z_k\}$ with $z_k < z$, compute their densities $d(z_k)$ and the corresponding alpha values $\alpha_k = 1 - \exp(-\mathrm{ReLU}(d_k))$, and finally get visibility $v(z) = \prod_{k=1}^{K_r}(1 - \alpha_k)$. Though the formulation is a valid visibility function, it is computationally impractical because it requires K_r times evaluation of ϕ to compute the visibility of a input view to a point.

3.4. Generalize with NeuRay

When rendering in unseen scenes, we extract the visibility feature maps $\boldsymbol{G}$ from a cost volume construction [60] or a patch-match stereo [47]. In the cost volume construction on every input view, we use its N_s neighboring input views to construct a cost volume of size $H \times W \times D$ [60]. Then, a CNN is applied on the cost volume to produce the visibility feature map $\boldsymbol{G} \in \mathbb{R}^{H \times W \times C}$ for this input view. Alternatively, we can also directly extract feature maps $\boldsymbol{G}$ from depth maps estimated by patch-match stereo [47], in which the estimated depth map of size $H \times W$ is processed by a CNN to produce the visibility feature map $\boldsymbol{G}$.

Pipeline. The whole pipeline of rendering with NeuRay in an unseen scene is shown in Fig. 4. On all input views, cost volumes or depth maps are estimated by MVS algorithms [47, 60], which are processed by a CNN to produce visibility feature maps $\boldsymbol{G}$. Then, for 3D sample points on test rays, we compute the visibility $v_{i,j}$ of input views to these points (Sec. 3.3) and aggregate the local features $\boldsymbol{f}_{i,j}$ along with $v_{i,j}$ to compute the alpha values and colors on these points (Sec. 3.2). Finally, the alpha values and colors are accumulated along the test rays by volume rendering to synthesize the test images (Sec. 3.1).

Loss. The whole rendering framework can be pretrained on training scenes and then directly applied on unseen scenes for rendering. To pretrain the rendering framework, we randomly select a view in a training scene as the test view and use other views as input views to render the selected test view with a render loss

$$\ell_{render} = \sum \|\boldsymbol{c} - \boldsymbol{c}_{gt}\|^2, \tag{6}$$

where $\boldsymbol{c}$ is computed by Eq. 1, $\boldsymbol{c}_{gt}$ is the ground-truth color.

3.5. Finetune with NeuRay

As done in previous works [5, 55, 56, 63], the proposed rendering framework can be further finetuned on a specific scene to achieve better rendering quality on this scene. In the given scene, we randomly select an input view as a pseudo test view and use the other input views to render the pseudo test view for training. Additionally, we add trainable parameters of NeuRay and a consistency loss in finetuning.

Trainable parameters of NeuRay. Fig. 5 shows the detailed structure of the CNN in Fig. 4. On every input view, an intermediate feature map $\boldsymbol{G}' \in \mathbb{R}^{H \times W \times C}$ between the constructed cost volume (or the estimated depth map) and the visibility feature map $\boldsymbol{G}$ is treated as trainable parameters of NeuRay for this input view. We call the convolution layers before the $\boldsymbol{G}'$ as an *initialization network* and the convolution layers between $\boldsymbol{G}'$ and $\boldsymbol{G}$ as a *visibility encoder*. Parameters $\boldsymbol{G}'$ are not trained from scratch but initialized by the initialization network using the constructed cost volume or the estimated depth map. Then, the initialization network is discarded while $\boldsymbol{G}'$ along with other network parameters are optimized in finetuning.

Discussion. Alternatively, we may directly make $\boldsymbol{G}$ as trainable parameters of NeuRay and discard both the initialization network and the visibility encoder. However, we find that making $\boldsymbol{G}'$ trainable and applying the visibility encoder on the trainable $\boldsymbol{G}'$ improve the visibility prediction. Because the convolution layers in the visibility encoder associate feature vectors of nearby pixels on $\boldsymbol{G}'$ and these nearby pixels usually have similar visibility.

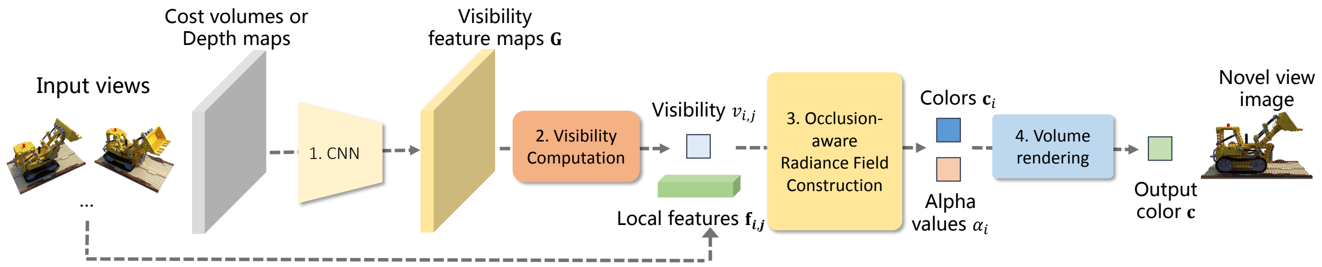
Input views
...
Cost volumes or Depth maps
1. CNN
Visibility feature maps G
2. Visibility Computation
Visibility $v_{i,j}$
Local features $\mathbf{f}_{i,j}$
3. Occlusion-aware Radiance Field Construction
Colors $\mathbf{c}_i$
Alpha values α_i
4. Volume rendering
Output color $\mathbf{c}$
Novel view image

Figure 4. Pipeline of rendering with NeuRay. 1. On input views, cost volumes or depth maps are estimated, which are used in predicting visibility feature maps by a CNN (Sec. 3.4). 2. Visibility feature maps are used in the computation of the visibility of input views to 3D points (Sec. 3.3). 3. For 3D points, we aggregate local features from input views along with the visibility to compute alpha values and colors on these points (Sec. 3.2). 4. Volume rendering is applied to accumulate alpha values and colors to synthesize images (Sec. 3.1).

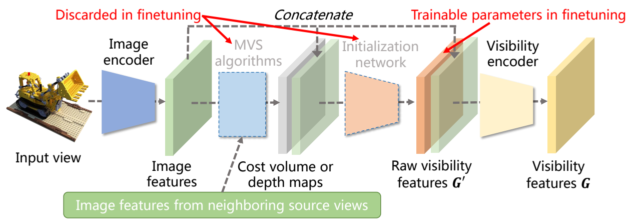

Figure 5. When finetuning on a scene, we treat the intermediate feature map $\boldsymbol{G}'$ as trainable parameters of NeuRay, which is initialized by the initialization network. Then, the initialization network will be discarded in finetuning.

Consistency loss. Besides the rendering loss, we additionally use a consistency loss in finetuning. Since both the constructed radiance field and the visibility of NeuRay depict the scene geometry, we can enforce the consistency between them in finetuning. Specifically, in finetuning, we sample points $\boldsymbol{p}_i \equiv \boldsymbol{p}(z_i)$ on a pseudo test ray and compute the hitting probability h_i on the sample points from the constructed radiance field. Meanwhile, the pseudo test view is also an input view, on which there is a NeuRay representation to decode a distribution $t(z)$ for this pseudo test ray. Based on $t(z)$, we compute a new hitting probability $\tilde{h}_i$ on every sample point $\boldsymbol{p}_i$ by

$$\tilde{h}_i = t(z_{i+1}) - t(z_i), \tag{7}$$

where z_i is the depth of the point. Thus, we can enforce the consistency between $\tilde{h}_i$ and h_i to construct a loss

$$\ell_{consist} = \frac{1}{K_t}\sum_{i=1}^{K_t} CE(\tilde{h}_i, h_i), \tag{8}$$

where CE is the cross entropy loss.

Discussion. Similar to previous generalization methods [5, 55, 56, 63], finetuning on a specific scene refines our network parameters for better feature aggregation and thus better radiance field construction on the scene. Meanwhile, further adding the trainable parameters of NeuRay and the consistency loss enables our rendering framework to refine the NeuRay representation, which brings better occlusion inference and significantly improves the rendering quality. This actually enables a memorization mechanism as illustrated in the supplementary material.

3.6. Speeding up rendering with NeuRay

On a test ray, most sample points $\boldsymbol{p}_i$ have nearly zero hitting probabilities h_i and thus do not affect the output color. However, most computations are wasted on feature aggregation on these empty points. In the following, we show that a coarse hitting probability $\hat{h}_i$ on a test ray can be directly computed from NeuRay using little computation. Then, only very few fine points are sampled around points with large $\hat{h}_i$ for feature aggregation.

To compute the hitting probability $\hat{h}_i$ defined on the **test ray**, we first define the alpha value $\tilde{\alpha}$ in depth range (z_0, z_1) on an **input ray** by

$$\tilde{\alpha}(z_0, z_1) = \frac{t(z_1) - t(z_0)}{1 - t(z_0)}. \tag{9}$$

Then, we compute the $\hat{h}_i$ on a sample point $\boldsymbol{p}_i$ by

$$\hat{\alpha}_i = \frac{\sum_j \tilde{\alpha}_{i,j}(z_{i,j}, z_{i,j} + l_i) v_{i,j}}{\sum_j v_{i,j}}, \tag{10}$$

$$\hat{h}_i = \prod_{k=1}^{i-1}(1 - \hat{\alpha}_k)\hat{\alpha}_i, \tag{11}$$

where $z_{i,j}$ is the depth of the point on j-th input view, $l_i = z_{i+1} - z_i$ is the distance between the point $\boldsymbol{p}_i$ and its subsequent point $\boldsymbol{p}_{i+1}$ on the test ray. Computation of $\hat{h}$ is very fast because it only involves a simple combination of $t(z)$ on input views. We discuss the rationale behind the design of $\tilde{\alpha}$, $\hat{\alpha}$ and $\hat{h}$ and their connections to NeRF [31]-style density in the supplementary material.

Settings	Method	Synthetic Object NeRF PSNR↑	SSIM↑	LPIPS↓	Real Object DTU PSNR↑	SSIM↑	LPIPS↓	Real Forward-facing LLFF PSNR↑	SSIM↑	LPIPS↓
Generalization	PixelNeRF [63]	22.65	0.808	0.202	19.40	0.463	0.447	18.66	0.588	0.463
	MVSNeRF [4]	25.15	0.853	0.159	23.83	0.723	0.286	21.18	0.691	0.301
	IBRNet [56]	26.73	0.908	0.101	25.76	0.861	0.173	25.17	0.813	0.200
	Ours	**28.29**	**0.927**	**0.080**	**26.47**	**0.875**	**0.158**	**25.35**	**0.818**	**0.198**
Finetuning	MVSNeRF [4]	27.21	0.888	0.162	25.41	0.767	0.275	23.54	0.733	0.317
	NeRF [31]	31.01	0.947	0.081	28.11	0.860	0.207	26.74	0.840	0.178
	IBRNet [56]	30.05	0.935	0.066	29.17	0.908	0.128	26.87	0.848	0.175
	Ours	**32.35**	**0.960**	**0.048**	**29.79**	**0.928**	**0.107**	**27.06**	**0.850**	**0.172**

Table 1. Quantitative comparison with baseline methods.

4. Experiment

4.1. Experimental Protocols

4.1.1 Datasets

We use two kinds of evaluation datasets, the object dataset and the forward-facing dataset. The results are evaluated by PSNR, SSIM [17] and LPIPS [64] as the metrics.

Object dataset. The object dataset includes the NeRF synthetic dataset [34] and the DTU dataset [18]. The NeRF synthetic dataset has 8 objects, each of which contains 100 images as input views and the other 200 images as test views. For the DTU dataset, we select 4 objects (birds, tools, bricks and snowman) as test objects. On each test object, we leave out 1/8 images as test views and the rest images as input views.

Forward-facing dataset. The forward-facing dataset is the LLFF dataset [30] with 8 scenes. Each scene contains 20 to 62 images. We follow the same train-test set split for each scene as previous methods [31, 56], which uses 1/8 images as test views. As done in [57], all images are undistorted by COLMAP [46]. The evaluation resolution are 1008×756 for the LLFF dataset, 800×800 for the NeRF synthetic dataset and 800×600 for the DTU dataset. The test images in two object datasets all use black backgrounds.

Training dataset. In order to train the generalization model, we use three kinds of datasets: (1) the synthetic Google Scanned Object dataset [10], which contains 1023 objects with 250 rendered images on each object; (2) three forward-facing training datasets [8, 30, 65] and (3) the rest training objects from the DTU dataset.

4.1.2 Implementation details

To render a test view, we do not use all input views but $N_w = 8$ neighboring input views, called **working views**. $D = 64$ planes are used in the cost volume while we use COLMAP [47] for patch-match stereo. $N_l = 2$ logistics distributions are mixed in $t(z)$. We use the coarse-to-fine sampling strategy as done in [31, 56] with 64 sample points in both stages. Coarse and fine models share the same image encoder, visibility encoder and initialization network, but they use different decoder $\mathcal{F}$ and aggregation networks. The aggregation networks follows similar design as [56] but with additional visibility as inputs, which blends input colors and applies a transformer along test rays. All experiments are conducted on a 2080 Ti GPU. Details and architectures can be found in the supplementary material.

4.2. Comparison with baselines

Experiment settings. We compare with IBRNet [56], PixelNeRF [63], MVSNeRF [4] and NeRF [31] in the generalization setting and the finetuning setting. In the generalization setting, all generalization methods including our method are pretrained on the same training scenes and tested on unseen test scenes. In the finetuning setting, all generalization methods including ours are further finetuned on the input views of each test scene while NeRF [31] is trained-from-scratch.

The quantitative results are shown in Table. 1 and the qualitative results are shown in Fig. 6. Table 1 shows that our method generalizes well to unseen scenes and outperforms all other generalization models. After scene-specific finetuning, our method clearly outperforms all baselines on two object datasets but achieves similar performance as IBRNet [56] on the LLFF dataset [30]. The reason is that the LLFF dataset contains very dense forward-facing input views so that every 3D point is visible to a large number of input views. In this case, even one or two views are occluded, there are still enough visible views to provide feature consistency for IBRNet [56] to render correctly. In comparison, in the Synthetic NeRF dataset, images are sparsely captured around the object in 360°, which brings more severe feature inconsistency to reduce the performance of IBRNet [56]. In contrast, our rendering is occlusion-aware so that our model performs much better on the NeRF synthetic dataset. Furthermore, we show that our method outperforms IBRNet by a large margin with sparse working views on the LLFF dataset in the supplementary.

By comparing MVSNeRF [4] with our method in Fig. 6, we notice that finetuning MVSNeRF [4] leads to noisy arti-

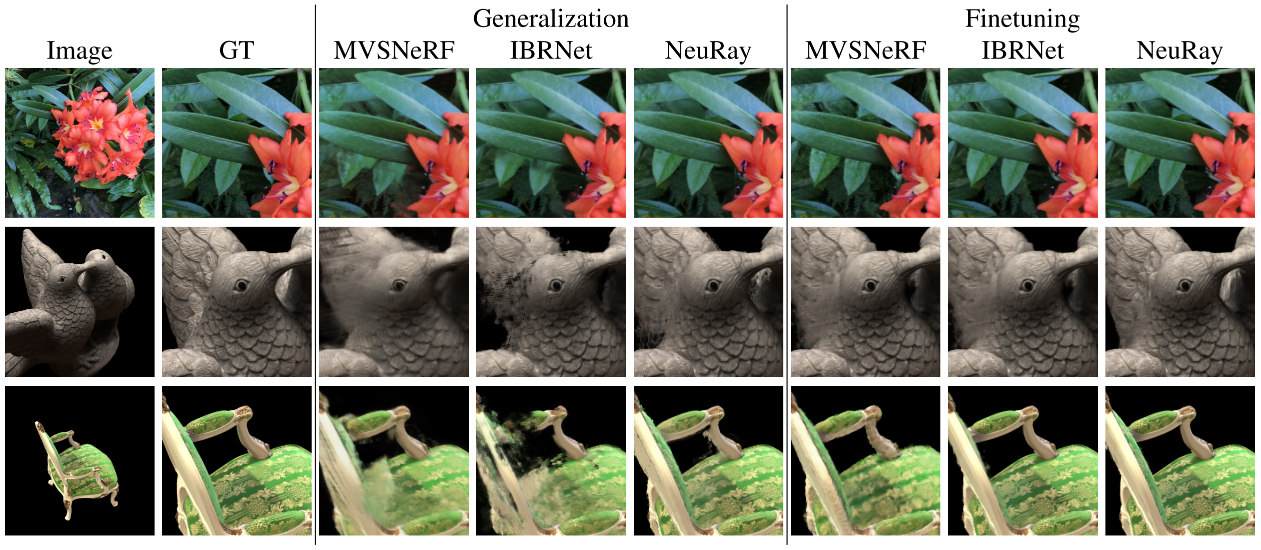

Figure 6. Qualitative results of different methods. Please refer to the supplementary materials for more results.

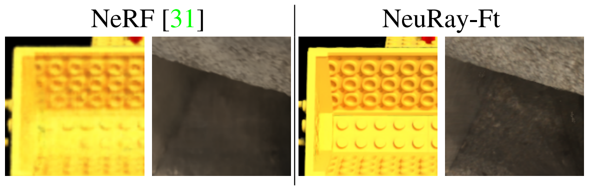

Figure 7. Our method renders details more clearly than NeRF [31].

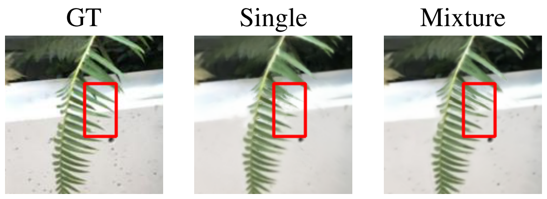

Figure 8. Comparison between the mixture logistics distribution $N_l = 2$ and the single logistics distribution $N_l = 1$ as the occlusion probability. Mixture logistics distribution improves the results on the edges with abrupt depth changes (red rectangle).

ID	Description	Setting	Lego	Fern
1	IBRNet	Gen	25.64	24.16
2	Only aggregation	Gen	25.61	22.25
3	Aggregation with depth features	Gen	26.45	22.43
4	Aggregation with init-NeuRay	Gen	28.41	24.02
5	mixture logistics $N_l = 2$	Ft	32.97	25.93
6	mixture logistics $N_l = 2$	Sc	33.07	25.89
7	single logistics $N_l = 1$	Sc	33.05	25.58
8	Only aggregation	Sc	29.61	24.40
9	NeuRay without $\ell_{consist}$	Sc	31.46	25.24

Table 2. Ablation studies. PSNRs on the "Lego" from the NeRF synthetic dataset and the "Fern" from the LLFF dataset are reported. "Gen" means the generalization setting, "Ft" means finetuning on the scene and "Sc" means training from scratch.

facts when the test view is far from the input view on which the cost volume is built. By comparing NeRF [30] with our method in Fig. 7, we find that NeRF needs more optimization steps to recover subtle details like surfaces of the bricks and textures on the Lego while it is relatively more easy for our method to render these details by blending colors of input views, which is the reason that our method can achieve better rendering quality than NeRF.

4.3. Ablation studies

How effective is the visibility from init-NeuRay? To validate this, in Table 2, we test the IBRNet [56] (ID 1), the model with only image feature aggregation (ID 2), the model aggregating images features with estimated depth from COLMAP [47] (ID 3) and the model aggregating image features with visibility of NeuRay initialized by cost volumes (ID 4). Since our image encoder is shallower than IBRNet, the performance with only image feature aggregation is worse than IBRNet. Simply adding estimated depth only brings slight improvement while using visibility of initialized NeuRay significantly improves the quality.

Can NeuRay be trained from scratch? To show NeuRay can be constructed on a scene from scratch without initialization from cost volumes, we follow exactly the same process as finetuning to train our method but the raw visibility feature maps $\boldsymbol{G}'$ and parameters of all networks are randomly initialized. Results in Table 2 (ID 5 and 6) show that training our method from scratch is also able to achieve similar results as finetuning the initialized pretrain model.

Single or mixture logistics distributions? As discussed in Sec. 3.3, the choice of $t(z)$ can be a single logistics distribution or a mixture of logistics distributions. We compare these two choices in Table 2 (ID 6 and 7) and show quali-

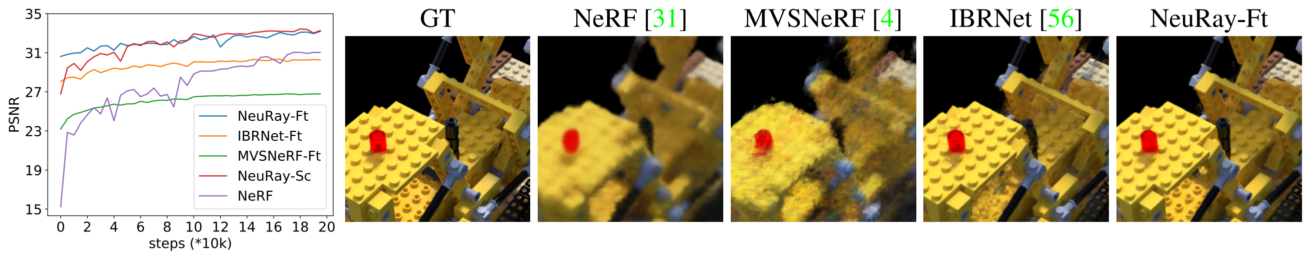

Figure 9. (Left) Curves of PSNR of different models with different training steps. PSNR is computed on a validation set of the "Lego". (Right) Qualitative results of models with 10k training steps. Due to the limitation of GPU memory, the batch size for IBRNet [56] and our method is 512 while the batch size for MVSNeRF [4] and NeRF [31] is 1024.

Method	$\hat{h}$	$K_{t,1}$	$K_{t,2}$	PSNR	Time(s)
IBRNet	✗	64	64	34.00	31.46
NeRF	✗	128	128	33.65	31.51
NeuRay	✗	64	64	35.33	30.03
NeuRay	✓	64	8	34.57	3.95
NeuRay	✓	32	4	33.73	2.57

Table 3. Rendering time and PSNR of the "birds" from the DTU dataset. $K_{t,1}$ and $K_{t,2}$ are numbers of points used in coarse and fine sampling respectively, $\hat{h}$ means using the probability $\hat{h}$ to conduct coarse sampling, "Time" means the time cost on rendering one 800×600 images on a 2080Ti GPU.

Method	Train Step	Train Time	PSNR
NeRF [31]	200k	∼9.5h	30.27
MVSNeRF [4]-Ft	10k	∼28min	23.77
IBRNet [56]-Ft	5k	∼41min	28.38
NeuRay-Ft	5k	∼32min	30.63

Table 4. PSNR and training steps/time on NeRF synthetic dataset.

tative comparison in Fig. 8, which demonstrates that using a mixture of logistics distributions improves the rendering quality in regions with abrupt depth changes.

How effective is NeuRay in per-scene optimization? In Table 2, we compare three models, the full model (ID 6), the model with only image feature aggregation (ID 8) and the model with NeuRay but without $\ell_{consist}$ (ID 9). The results show that adding a NeuRay as a backend in per-scene optimization already brings improvements and further adding $\ell_{consist}$ enables memorization of geometry thus greatly improves the rendering quality.

4.4. Analysis

Speed up rendering with NeuRay. As discussed in Sec. 3.6, we can efficiently estimate a coarse $\hat{h}$ from the NeuRay to perform the coarse sampling and only sample few fine points based on the coarse sampling. To validate this, we conduct an experiment on the "birds" from the DTU dataset [18]. As shown in Table 3, only 4 or 8 subsequent sample points are enough for our method to achieve high-quality renderings, which speeds up the rendering 10 times from 30s to ∼3s. Further speeding up by baking out a mesh or occupancy voxels from alpha values $\hat{e}$ is possible, which we leave for future works.

Convergence speed. To show how the rendering quality of different models improves in the scene-specific optimization process, we train different models on the "Lego" and plot the curves of PSNR on a small validation set in Fig. 9 (left). The curves show that finetuning our method produces consistently better rendering results than training all baseline methods with the same training steps. In Fig. 9 (right), we show the qualitative results on 10k training steps. With only 10k training steps, NeRF [31] and MVSNeRF [4] are still far from convergence thus produce blurred images, IBRNet [56] produces artifacts on regions with occlusions while our method already produces high-quality renderings.

Only finetune few steps with NeuRay. Table 4 reports PSNR and time of different models with only few finetuning steps on the NeRF synthetic dataset. Note that finetuning both IBRNet [56] and our method requires image feature extraction which costs more time on one training step than neural fields MVSNeRF [4] and NeRF [31]. Since our image encoder is shallower than IBRNet, finetuning our method is slightly faster. The results show that our method is able to be finetuned limited time (32min) to achieve similar quality as NeRF [31] with long training time (9.5h), which is significantly better than the other generalization methods with similar finetuning time.

5. Conclusion

In this paper, we proposed a novel neural representation NeuRay for the novel view synthesis task. NeuRay represents a scene by occlusion probabilities defined on input rays and is able to efficiently estimate the visibility from arbitrary 3D points to input views. With the help of NeuRay, we are able to consider the visibility when constructing radiance fields by multi-view feature aggregation. Experiments on the DTU dataset, the NeRF synthetic dataset and the LLFF dataset demonstrate that our method can render high-quality images without any training on the scene or with only few finetuning steps on the scene.

References

[1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *ECCV*, 2020. 2

[2] Alexander W. Bergman, Petr Kellnhofer, and Gordon Wetzstein. Fast training of neural lumigraph representations using meta learning. In *NeurIPS*, 2021. 2

[3] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM TOG*, 2013. 2

[4] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. MVSNeRF: fast generalizable radiance field reconstruction from multi-view stereo. In *ICCV*, 2021. 2, 6, 8

[5] Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. Stereo radiance fields (SRF): Learning view synthesis for sparse views of novel scenes. In *CVPR*, 2021. 1, 2, 3, 4, 5

[6] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. Extreme view synthesis. In *ICCV*, 2019. 2

[7] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. In *Eurographics*, 2012. 2

[8] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *CVPR*, 2019. 6

[9] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-fidelity neural rendering at 200fps. In *ICCV*, 2021. 2

[10] Research Google. Google scanned objects. https://app.ignitionrobotics.org/GoogleResearch/fuel/collections/GoogleScannedObjects, 2021. 6

[11] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *SIGGRAPH*, 1996. 2

[12] Marc Habermann, Lingjie Liu, Weipeng Xu, Michael Zollhoefer, Gerard Pons-Moll, and Christian Theobalt. Real-time deep dynamic characters. *ACM TOG*, 2021. 2

[13] Tong He, John Collomosse, Hailin Jin, and Stefano Soatto. Geo-PIFu: Geometry and pixel aligned implicit functions for single-view human reconstruction. *arXiv preprint arXiv:2006.08072*, 2020. 2

[14] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM TOG*, 2018. 1, 2

[15] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *ACM TOG*, 2016. 2

[16] Yang Hong, Juyong Zhang, Boyi Jiang, Yudong Guo, Ligang Liu, and Hujun Bao. StereoPIFu: Depth aware clothed human digitization via stereo vision. In *CVPR*, 2021. 2

[17] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *CVPR*, 2010. 6

[18] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *CVPR*, 2014. 2, 6, 8

[19] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. Learning-based view synthesis for light field cameras. *ACM TOG*, 2016. 1, 2

[20] Petr Kellnhofer, Lars Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *CVPR*, 2021. 2

[21] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Computer Graphics Forum*, volume 40, pages 29–43. Wiley Online Library, 2021. 2

[22] 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. 2

[23] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. 2

[24] Lingjie Liu, Marc Habermann, Viktor Rudnev, Kripasindhu Sarkar, Jiatao Gu, and Christian Theobalt. Neural actor: Neural free-view synthesis of human actors with pose control. In *SIGGRAPH Asia*, 2021. 2

[25] Lingjie Liu, Weipeng Xu, Marc Habermann, Michael Zollhöfer, Florian Bernard, Hyeongwoo Kim, Wenping Wang, and Christian Theobalt. Neural human video rendering by learning dynamic textures and rendering-to-video translation. *IEEE TVCG*, 2020. 2

[26] Lingjie Liu, Weipeng Xu, Michael Zollhoefer, Hyeongwoo Kim, Florian Bernard, Marc Habermann, Wenping Wang, and Christian Theobalt. Neural rendering and reenactment of human actor videos. *ACM TOG*, 2019. 2

[27] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *CVPR*, 2020. 2

[28] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural Volumes: Learning dynamic renderable volumes from images. In *SIGGRAPH*, 2019. 2

[29] Ricardo Martin-Brualla, Noha Radwan, Mehdi Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the wild: Neural radiance fields for unconstrained photo collections. In *CVPR*, 2021. 2

[30] 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 TOG*, 2019. 1, 2, 6, 7

[31] 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, 5, 6, 7, 8

[32] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022. 2

[33] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021. 2

[34] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *CVPR*, 2020. 2, 6

[35] Sida Peng, Junting Dong, Qianqian Wang, Shangzhan Zhang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Animatable neural radiance fields for human body modeling. In *ICCV*, 2021. 2

[36] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Neural Body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *CVPR*, 2021. 2

[37] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM TOG*, 2017. 2

[38] Amit Raj, Michael Zollhoefer, Tomas Simon, Jason Saragih, Shunsuke Saito, James Hays, and Stephen Lombardi. PVA: Pixel-aligned volumetric avatars. In *CVPR*, 2021. 2

[39] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, 2021. 2

[40] Konstantinos Rematas, Ricardo Martin-Brualla, and Vittorio Ferrari. ShaRF: Shape-conditioned radiance fields from a single view. In *ICML*, 2021. 2

[41] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *ECCV*, 2020. 2

[42] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *CVPR*, 2021. 2

[43] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *arXiv preprint arXiv:2110.06635*, 2021. 2

[44] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *ICCV*, 2019. 2

[45] Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. PIFuHD: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *CVPR*, 2020. 2

[46] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 6

[47] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*, 2016. 1, 2, 4, 6, 7

[48] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. DeepVoxels: Learning persistent 3d feature embeddings. In *CVPR*, 2019. 2

[49] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *NeurIPS*, 2019. 2

[50] 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 *CVPR*, 2021. 2

[51] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *arXiv preprint arXiv:2111.11215*, 2021. 2

[52] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*, 2021. 2

[53] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM TOG*, 2019. 2

[54] Justus Thies, Michael Zollhöfer, Christian Theobalt, Marc Stamminger, and Matthias Nießner. IGNOR: image-guided neural object rendering. In *ICLR*, 2020. 2

[55] Alex Trevithick and Bo Yang. GRF: Learning a general radiance field for 3D scene representation and rendering. In *ICCV*, 2021. 1, 2, 3, 4, 5

[56] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. IBRNet: Learning multi-view image-based rendering. In *CVPR*, 2021. 1, 2, 3, 4, 5, 6, 7, 8

[57] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 2, 6

[58] Minye Wu, Yuehao Wang, Qiang Hu, and Jingyi Yu. Multi-view neural human rendering. In *CVPR*, 2020. 2

[59] Zexiang Xu, Sai Bi, Kalyan Sunkavalli, Sunil Hadap, Hao Su, and Ravi Ramamoorthi. Deep view synthesis from sparse photometric images. *ACM TOG*, 2019. 2

[60] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. MVSNet: Depth inference for unstructured multi-view stereo. In *ECCV*, 2018. 1, 2, 4

[61] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 2

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

[63] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 1, 2, 3, 4, 5, 6

[64] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

[65] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018. 6