

Neural Rays for Occlusion-aware Image-based Rendering

YUAN LIU, The University of Hong Kong, China

SIDA PENG, Zhejiang University, China

LINGJIE LIU, Max Planck Institute for Informatics, Germany

QIANQIAN WANG, Cornell University, U.S.A

PENG WANG, The University of Hong Kong, China

CHRISTIAN THEOBALT, Max Planck Institute for Informatics, Germany

XIAOWEI ZHOU, Zhejiang University, China

WENPING WANG, Texas A&M University, U.S.A

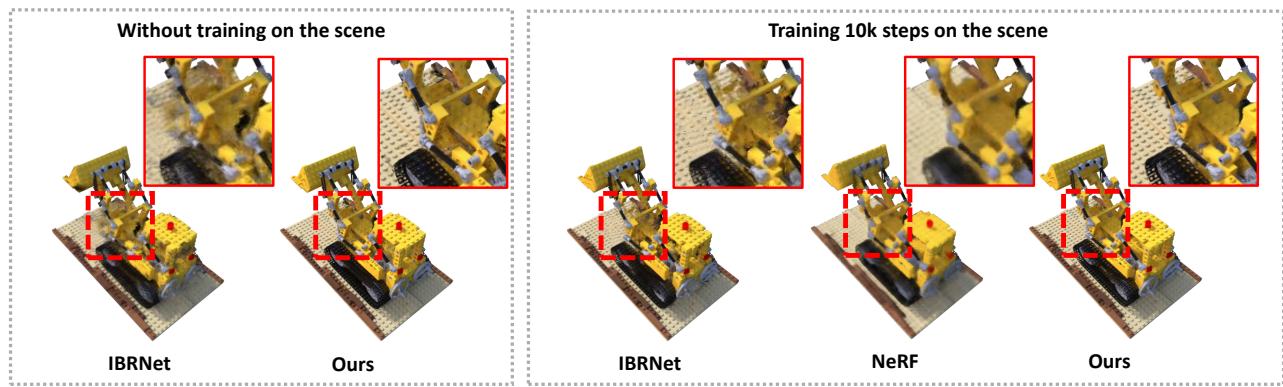


Fig. 1. Synthesized unseen views of different methods. We propose a novel neural ray representation for the novel view synthesis task. Given a set of input views, we construct an initial neural ray representation from these views in a single forward pass and render images of unseen poses from the representation. Furthermore, we can refine the neural ray representation by training it on the scene to achieve better renderings with only a few training steps.

We present a new neural representation, called *Neural Ray* (NeuRay), for the novel view synthesis (NVS) task with multi-view images as input. Existing neural scene representations for solving the NVS problem, such as NeRF [Mildenhall et al. 2020], cannot generalize to new scenes and take excessively long time on training on each new scene from scratch. The other subsequent neural rendering methods based on stereo matching, such as PixelNeRF [Yu et al. 2020], SRF [Chibane et al. 2021] and IBRNet [Wang et al. 2021b] are designed to generalize to unseen scenes but suffer from view inconsistency in complex scenes with self-occlusions. To address these issues, our NeuRay method represents every scene by encoding the visibility of rays associated with the input views. This neural representation can efficiently be initialized from depths estimated by external MVS methods, which is able to generalize to new scenes and achieves satisfactory rendering images without

Authors' addresses: Yuan Liu, The University of Hong Kong, China; Sida Peng, Zhejiang University, China; Lingjie Liu, Max Planck Institute for Informatics, Germany; Qianqian Wang, Cornell University, U.S.A; Peng Wang, The University of Hong Kong, China; Christian Theobalt, Max Planck Institute for Informatics, Germany; Xiaowei Zhou, Zhejiang University, China; Wenping Wang, Texas A&M University, U.S.A.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Association for Computing Machinery.

0730-0301/2021/7-ART \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

any training on the scene. Then, the initialized NeuRay can be further optimized on every scene with little training timing to enforce spatial coherence to ensure view consistency in the presence of severe self-occlusion. Experiments demonstrate that NeuRay can quickly generate high-quality novel view images of unseen scenes with little finetuning and can handle complex scenes with severe self-occlusions which previous methods struggle with.¹

CCS Concepts: • Computing methodologies → Image-based rendering.

Additional Key Words and Phrases: novel view synthesis, neural scene representation, image-based rendering

ACM Reference Format:

Yuan Liu, Sida Peng, Lingjie Liu, Qianqian Wang, Peng Wang, Christian Theobalt, Xiaowei Zhou, and Wenping Wang. 2021. Neural Rays for Occlusion-aware Image-based Rendering. *ACM Trans. Graph.* 1, 1 (July 2021), 16 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

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

¹Project page:<https://liuyuan-pal.github.io/NeuRay/>

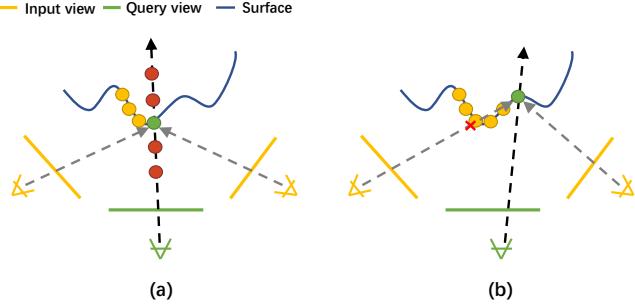


Fig. 2. Recent stereo matching based rendering methods (e.g. PixelNeRF, IBRNet, and SRF) suffer from view inconsistency in the presence of surface occlusions when using local features in input views to identify surfaces for novel view synthesis. (a) The case without surface occlusions: When rendering the green point, the features of the two rays from different input views are consistent on this point so that a large density value is correctly assigned to the green point. (b) The case with surface occlusions: When rendering the green point in (b), the ray from the left input view is occluded by the surface. The local features of the two rays are inconsistent on the green point due to the occlusion, which leads to erroneous results. Note that only two input views are shown here for illustration. In practice, there may be more input views, which makes this problem even more severe.

NVS problem compared to earlier image-based rendering methods [Hedman et al. 2018; Kalantari et al. 2016; Mildenhall et al. 2019]. Neural Radiance Fields (NeRF) [Mildenhall et al. 2020] and related works [Liu et al. 2020a; Reiser et al. 2021] show that photo-realistic images of unseen views can be synthesized from a 5D radiance field encoded in a neural network. This neural network learns a radiance field ($\sigma, c = f(p, r)$) which maps a position p and a direction r to a density σ and a color c . Novel views can be synthesized with a volume renderer. 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 like IBRNet [Wang et al. 2021b], Stereo Radiance Field (SRF) [Chibane et al. 2021] or PixelNeRF [Yu et al. 2020] aim at designing NeRF-like neural rendering frameworks that can generalize to unseen scenes. These works are motivated by traditional stereo matching methods which extract local image features on the input views and match these features to determine 3D surfaces, i.e., the density σ in these methods, on a specific 3D point, as shown in Fig. 2 (a). Then, the color of this point is either directly regressed from features of input views [Chibane et al. 2021; Yu et al. 2020] or computed as a weighted combination of colors of input views [Wang et al. 2021b]. While these methods can generalize to unseen scenes of forward-facing datasets with dense input views like the LLFF [Mildenhall et al. 2019] dataset, they struggle on datasets with fewer input views, more complex shapes and notable self-occlusions (e.g. the NeRF Synthetic dataset [Mildenhall et al. 2020]), even after finetuning on the scene. The reason is that these methods only assume local feature consistency and are oblivious of the global scene structure, which may produce erroneous renderings due to surface occlusions as illustrated in Fig. 2 (b).

To overcome this limitation, we present a novel 3D representation, called *Neural Ray* or *NeuRay* for short, for the novel view synthesis

task. The key idea of NeuRay is to represent the whole scene by so-called *reference rays*, which are camera rays emitted from the input views to the 3D space. On each reference ray, NeuRay encodes a probability function about how far this ray can reach before hitting a surface. This allows us to determine whether a 3D point is visible from this particular ray. When computing the density and the color of a 3D point p , NeuRay uses the encoded visibility to check whether the reference ray from each input view to the point p is occluded by other surfaces in the scene, and uses only the colors or features of those rays that are not occluded. Such self-occlusion checking greatly enhances rendering quality on difficult scenes, as demonstrated by experimental results.

To compute the visibility for occlusion inference, existing neural rendering methods, such as NeRF [Mildenhall et al. 2020] and its variants [Liu et al. 2020a], need to accumulate volume densities along the ray, which is time-consuming and inefficient for fast rendering. In contrast, NeuRay directly encodes ray visibility on each reference ray, and thus achieves efficient occlusion testing and fast rendering with view consistency at test time.

Another advantage of representing the scene with reference rays is that the associated visibility probabilities can be easily initialized with estimated depth maps of the scene from some multi-view stereo algorithms. Then, we can quickly finetune the initialized NeuRay representation to achieve photo-realistic renderings of the quality comparable to state-of-the-arts. In contrast, fully implicit representations, such as NeRF, must take excessively long time to be trained from scratch on each specific scene.

At test time, given a constructed NeuRay representation and a pixel on the query image, we use volume rendering to compute the color of the pixel. This is done by first sampling points on the ray emitted from the pixel, then computing the colors and alpha values of the sample points, and finally accumulating the colors of the sample points to produce the pixel color. To compute the colors and alpha values of the sample points from NeuRay representation, we provide two methods. In the first method, called *Direct Rendering* (DR) of NeuRay, the colors and alpha values of each sample point are computed by directly combining the information of all the reference rays going through the sample point, using their geometric and visibility relationship. In the second method, called *Network Rendering* (NR), NeuRay uses trainable networks to combine multi-view features based on corresponding ray visibility to get the alpha values and colors of the sample points, which is similar to the stereo matching based rendering methods. The network rendering mode further improves the rendering quality of the direct rendering but at the cost of more computation and using more parameters than the direct rendering. In the end, we combine the efficiency of DR with the good rendering quality of NR by first applying DR once to coarsely estimate the alpha values of all sample points and only feed these points with large alpha values to NR, which achieves high-quality rendering in an efficient manner.

We conducted extensive experiments on three datasets, the NeRF synthetic dataset [Mildenhall et al. 2020], the DTU dataset [Jensen et al. 2014] and the forward-facing LLFF dataset [Mildenhall et al. 2019], to demonstrate the effectiveness of NeuRay. The results show that (1) NeuRay can be trained from scratch to achieve comparable or even better rendering results than NeRF; (2) NeuRay without

finetuning already produces satisfactory renderings that are superior to the results of IBRNet or PixelNeRF in the same generalization setting; and (3) NeuRay can further be finetuned quickly using just a few steps on a new scene to get better results than the NeRF trained from scratch with the same steps or the IBRNet finetuned with the same steps. Moreover, we also show that rendering a novel view of 400×400 resolution with NeuRay costs ~ 1 second and has the potential for further speeding up.

2 RELATED WORKS

2.1 Image-based rendering

Many works [Chaurasia et al. 2013; Gortler et al. 1996; Hedman et al. 2018; Kalantari et al. 2016; Levoy and Hanrahan 1996; Riegler and Koltun 2020] have focused on blending input images on scene geometry proxies to synthesize novel views, without recovering the detailed 3D geometry. In conventional light field-based methods [Davis et al. 2012; Gortler et al. 1996] sample and reconstruct a 4D plenoptic function from densely sampled views, which can achieve photo-realistic rendering results. However, these methods typically have a limited range of renderable viewpoints. To extend the renderable range, some works [Chaurasia et al. 2013; Penner and Zhang 2017] seek the help of 3D proxy geometry. Specifically, they mostly utilize multi-view stereo methods [Schönberger et al. 2016; Schönberger et al. 2016] to infer depth maps of input images, which are then used to warp input images to the target view. Finally, they perform image blending to the warped images to obtain the target image. With the development of deep learning techniques, some methods [Choi et al. 2019; Hedman et al. 2018; Kalantari et al. 2016; Riegler and Koltun 2020, 2021; Thies et al. 2020; Xu et al. 2019] introduce convolutional neural networks (CNNs) to replace hand-crafted components of the image-based rendering pipeline. DeepBlending [Hedman et al. 2018] leverages CNNs to predict blending weights of warped images for composition. To reduce the number of input views, [Xu et al. 2019] utilizes the advances of learning-based multi-view stereo methods to predict depth maps. IGNOR [Thies et al. 2020] proposes a network to model view-dependent effects and learns the network in a self-supervised manner. One core challenge for the image-based rendering methods is that they tend to be sensitive to the quality of estimated depth maps. To overcome this problem, [Choi et al. 2019] additionally infers the depth uncertainty and refines the synthesized image with the depth distribution. NeuRay also belongs to the category of image-based rendering, which can actually be regarded as a plenoptic function for the scene. Meanwhile, the initialization of NeuRay also takes the depth estimated by MVS algorithms as inputs. However, unlike existing methods heavily relying on the estimated depth, NeuRay itself can be trained from scratch on the scene without input depth or finetuned to correct errors of input depths.

2.2 Neural scene representation

Explicit representation. Recently, instead of estimating 3D proxy geometries explicitly, some methods have attempted to reconstruct a learned 3D representation from input images with a differentiable renderer, such as voxels [Lombardi et al. 2019; Sitzmann et al. 2019a], textured meshes [Habermann et al. 2021; Liu et al. 2020b, 2019; Thies

et al. 2019], and point clouds [Aliev et al. 2020; Wu et al. 2020]. For a specific scene, DeepVoxels [Sitzmann et al. 2019a] condenses input images into a persistent feature volume. To synthesize images, it projects the feature volume onto 2D feature maps of the query views, which are then interpreted into images based on image-to-image translation methods [Isola et al. 2017]. Neural Volumes [Lombardi et al. 2019] encodes input images into a global latent vector and decodes the global vector into a RGB-*alpha* volume, which can be directly rendered into images with volume rendering techniques. To improve the performance of the rendered meshes, [Thies et al. 2019] augments meshes with neural textures, which are learned jointly with a 2D neural renderer, enabling it to achieve impressive rendering results from inaccurate geometries. In similar spirit, neural point-based graphics [Aliev et al. 2020; Wu et al. 2020] augment point clouds by assigning 3D features to each 3D point. They then project 3D features onto 2D feature maps, which are translated into images with a U-Net based renderer.

Implicit representation. To improve the rendering resolution, recent methods [Kellnhofer et al. 2021; Liu et al. 2020a,c; Mildenhall et al. 2020; Niemeyer et al. 2020; Peng et al. 2021b; Sitzmann et al. 2019b] have adopted implicit functions to represent 3D scenes. As a pioneering work, SRN [Sitzmann et al. 2019b] proposes an MLP network that maps arbitrary 3D points to feature vectors, which is combined with a differentiable ray-marching algorithm to find intersections with the underlying geometry. On the intersection points, SRN uses another MLP network to map their feature vectors to colors. DVR [Niemeyer et al. 2020] represents the 3D scene with a continuous occupancy field and presents a root-finding algorithm to find surface points. To enable end-to-end training, it analytically derives the gradients of surface points using the implicit differentiation. Different from these methods, NeuRay represents the scene by explicit reference rays but associate every ray with a feature vector to implicitly encode the visibility probability. PIFu [Saito et al. 2019] and its follow-ups [He et al. 2020; Hong et al. 2021; Saito et al. 2020] adopt pixel-aligned implicit functions of occupancy fields to reconstruct human shapes, which can also be regarded as reference rays. The difference is that NeuRay is used in the NVS task and represents the geometry of reference rays by probability distributions.

NeRF and its variants. Recently, NeRF [Mildenhall et al. 2020] has shown photo-realistic rendering results by representing the scene as volume density and color fields, which works particularly well with volume rendering techniques. The following works [Garbin et al. 2021; Liu et al. 2020a, 2021; Martin-Brualla et al. 2020; Niemeyer and Geiger 2020; Peng et al. 2021a; Reiser et al. 2021; Wizadwongsu et al. 2021; Yu et al. 2021] have attempted to improve NeRF in various aspects, such as dynamic scene rendering and fast inference. One limitation of NeRF is that it needs to take a long time to be separately optimized on each new scene.

To overcome this limitation, recent works [Chibane et al. 2021; Raj et al. 2021; Rematas et al. 2021; Trevithick and Yang 2020; Wang et al. 2021b; Yu et al. 2020] introduce an image encoder for NeRF. For each 3D point, these methods first extract multi-view features from input images and then perform local stereo matching to aggregate features. Finally, they compute colors and densities based on aggregated features. Some recent works [Bergman et al. 2021; Tancik et al. 2021] resort to meta-learning to initialize neural representations for

fast training. NeuRay also adopts the volume rendering approach like NeRF and uses local stereo matching in the pipeline. However, unlike NeRF, NeuRay can be initialized from estimated depths to instantly produce a reasonably good neural representation for fast rendering. In contrast to stereo matching based methods which only rely on local feature consistency, NeuRay encodes the visibility distribution of the rays for the occlusion inference to significantly improve the rendering results.

3 METHOD

Given a set of input views of a scene, called *reference views*, with known camera poses, our goal is to render images from arbitrary novel views, called *query views*. Our key idea is to first represent the scene with reference rays emitted from reference views and then render query views based on these reference rays. The pipeline is shown by Fig. 3.

In the following, we first review related basics about volume rendering in Sec. 3.1 and describe how to represent a scene with NeuRay in Sec. 3.2. Then, we introduce the Direct Rendering (DR) of NeuRay in Sec. 3.3 and how to train NeuRay from scratch with DR in Sec. 3.4. Next, the Network Rendering (NR) and the scene-specific training with NR are introduced in Sec. 3.5 and Sec. 3.6 respectively. Finally, the initialization and the finetuning of NeuRay representation are discussed in Sec. 3.7.

3.1 Volume Rendering

A ray is parameterized by $\mathbf{p}(z) = \mathbf{o} + z\mathbf{r}$, $z \in \mathbb{R}^+$, where \mathbf{o} is the start point at the camera center, and \mathbf{r} is the unit direction vector of the ray. Given a query ray $\mathbf{p}(z)$ passing through a pixel of a query view, we sample K points $\{\mathbf{p}_i \equiv \mathbf{p}(z_i) | i = 1, \dots, K\}$ with the increasing values z_i along the ray, and we define $z_{K+1} = +\infty$. In the rendering process, following the ray casting approach in volume rendering, the color for the associated pixel is computed by

$$\mathbf{c}_o = \sum_{i=1}^K c_i h_i, \quad (1)$$

where $\mathbf{c}_o \in \mathbb{R}^3$ is the output color, c_i the color of the sample point \mathbf{p}_i , and h_i 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

$$h_i = \prod_{k=1}^{i-1} (1 - e_k) e_i, \quad (2)$$

where e_i is the alpha value in the depth range (z_i, z_{i+1}) . Next, we will introduce our NeuRay representation and how to compute the alpha value e_i and the color c_i from the NeuRay representation.

3.2 NeuRay representation

Given a reference view, we consider the rays emitted from all the pixels of the image, which we call *reference rays*. For each reference ray, we use $\mathbf{c} \in \mathbb{R}^3$ to denote the color of its associated pixel, and assign a trainable feature vector $\mathbf{f} \in \mathbb{R}^d$ to each ray to encode how far this ray can reach before hitting a surface. Specifically, based on the feature vector \mathbf{f} , we use an MLP network to predict the

probability of this ray being occluded before a certain depth z , as illustrated in Fig. 4. The **occlusion probability** $t(z)$ is given by

$$t(z; \alpha, \mu, \sigma) = \alpha S(z; \mu, \sigma), \quad (3)$$

where $\alpha \in [0, 1]$ is the total probability that this ray hits a surface, with $\alpha = 0$ meaning that the ray does not hit any surfaces. Here, $S(x; \mu, \sigma) = 1/(1 + e^{-(x-\mu)/\sigma})$ is the cumulative density function of a logistic distribution, μ is its mean, and $1/\sigma$ is its standard deviation. We compute the parameters α , μ and σ based on the feature vector \mathbf{f} using an MLP network ϕ , that this,

$$[\alpha, \mu, \sigma] = \phi(\mathbf{f}). \quad (4)$$

To render an image from a NeuRay representation, we define and compute the following two probabilities based on the occlusion probability, namely the **visibility probability** $v(z)$, the **hitting probability** $h(z_0, z_1)$, which are defined by

$$v(z) = 1 - t(z), \quad (5)$$

$$h(z_0, z_1) = t(z_1) - t(z_0), \quad (6)$$

Here the visibility probability $v(z)$ is the probability that the ray is visible at depth z , the hitting probability $h(z_0, z_1)$ is the probability that the ray is not occluded by any depth before z_0 and hits a surface in the depth range (z_0, z_1) , which is similar to the hitting probability h_i defined on the query ray in Eq. 1 and Eq. 2.

Alpha values on reference rays. Given the hitting probability $h(z_0, z_1)$, similar to Eq. 2, we can derive the corresponding alpha values $e(z_0, z_1)$ for reference rays, which is

$$e(z_0, z_1) = h(z_0, z_1) / v(z_0). \quad (7)$$

The following property can be proved.

PROPERTY 1. *Given a reference ray, suppose we sample the ray at the depths $\{z_i | i = 1, \dots, K\}$ on the ray. Then $e(\cdot, \cdot)$ defined in Eq. 7 and $h(\cdot, \cdot)$ defined in Eq. 6 satisfy $h(z_i, z_{i+1}) = \prod_{k=1}^{i-1} (1 - e(z_k, z_{k+1})) e(z_i, z_{i+1})$.*

We leave the proof in Appendix A.1. The derivation of alpha values from our defined hitting probability is inspired by NeuS [Wang et al. 2021a] which derives an *opacity value* from the defined unbiased weights. The difference is that opacity value in NeuS is based on signed distance function while the alpha value here is directly defined on the depth of a ray. The above-defined alpha values, hitting probability and visibility probability will be used in the rendering of a NeuRay representation. Additional discussion about the motivation of defining these probabilities can be found in Appendix A.2.

Choice of distribution *S.* Actually, any cumulative probability density function can be used as the function S . We choose the logistics distribution by default because it is unimodal thus suitable for modeling the hitting probability distribution on non-transparent objects. Meanwhile, to make such a distribution more flexible, we can adopt a mixture of multiple logistics distributions to achieve better performance when the scene is a clutter of objects or contains semi-transparent object, as validated in Sec. 4.5.6.

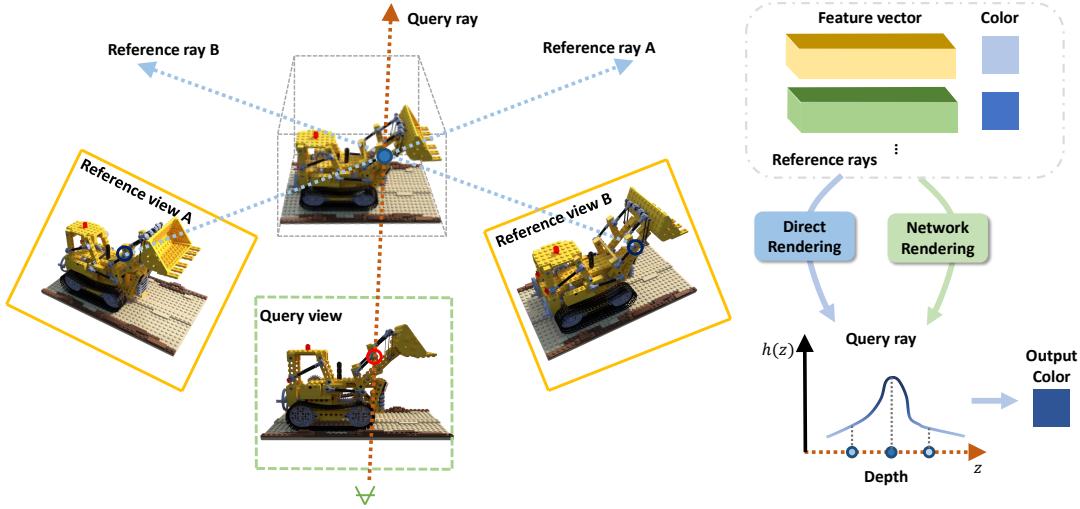


Fig. 3. Overview. In NeuRay, we represent the scene by reference rays emitted from the reference views to the 3D space. Every reference ray consists of a color and a feature vector that encodes how far this ray can reach before hitting a surface. Given a query ray, we sample points on it and provide two methods, namely *direct rendering* (DR) and *network rendering* (NR), for computing the hitting probabilities and colors of these sample points based on related reference rays. Finally, the hitting probabilities and colors of the sample points are accumulated along the query ray to compute the output color.

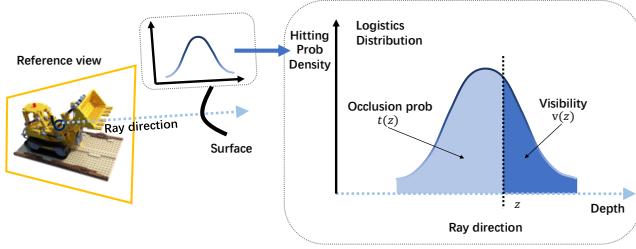


Fig. 4. Different probabilities defined on a reference ray. Note the figure uses probability density function (PDF) to illustrate these values while Eq. 3, Eq. 5 and Eq. 6 are defined with cumulative density function (CDF).

3.3 Direct rendering

Computation of e_i . For a sample point p_i on a query ray, we project it onto every reference view and find the reference rays from every reference view to this point. Denote the depth of p_i on the j -th reference view as $z_{i,j}$. Meanwhile, on each reference ray, we use Eq. 5 and Eq. 7 to compute the visibility denoted as $v_{i,j}$ and $e_{i,j}$, respectively. Intuitively, we take the alpha value e_i on the query ray as a weighted average of the alpha values $\{e_{i,j}\}$ of all related reference rays, which is

$$e_i = \frac{\sum_j e_{i,j}(z_{i,j}, z_{i,j} + l_i)v_{i,j}(z_{i,j})}{\sum_j v_{i,j}(z_{i,j})}, \quad (8)$$

where $l_i = z_{i+1} - z_i$ is the distance between the point p_i and its subsequent point p_{i+1} . Note that using the visibilities as weights enables NeuRay to be aware of occlusions, so that invisible reference rays do not contribute to the output e_i . To provide a better understanding of Eq. 8, we discuss several detailed cases in Appendix A.3.

Computation of c_i . To compute the color c_i of the point p_i in the direction r , we use a set of spherical harmonic functions as basis functions to fit a color function $R : \mathbb{S}^2 \rightarrow \mathbb{R}^3$ on every point. Specifically, we solve the following linear least squares problem

$$\min_{\theta_i} \sum_j h_{i,j}(z_{i,j}, z_{i,j} + l_i) \|R(r_{i,j}; \theta_i) - c_{i,j}\|^2 + \theta_i^\top \Lambda \theta_i, \quad (9)$$

where $h_{i,j}$ is the hitting probability in Eq. 6 of point p_i on the reference ray emitted from the j -th reference view, R the color function, θ_i the coefficients of spherical harmonic basis functions on this point, $r_{i,j}$ and $c_{i,j}$ are the viewing direction and color of the reference ray emitted from the j -th view, respectively, $\theta_i^\top \Lambda \theta_i$ a regularization term, and Λ a predefined or trainable diagonal matrix. This linear least squares problem has a closed-form solution as will be explained in Appendix A.4. After finding the solution θ_i to Problem 9, the color c_i is computed by

$$c_i = R(r; \theta_i). \quad (10)$$

Note that in Problem 9, the color difference is weighted by hitting probabilities so that occluded reference rays will not interfere the output colors. Meanwhile, since we fit a function R on the sphere, NeuRay is able to represent anisotropic colors (or radiance) at a point. NeX [Wizadwongsu et al. 2021] and PlencOctree [Yu et al. 2021], also use basis functions to represent colors. NeuRay directly finds the coefficients of basis functions by fitting while NeX or PlencOctree apply neural network to produce these coefficients.

Feature vector f . To produce the feature vector f associated with a reference ray, which is used in Eq. 4, we first place a set of trainable parameters $\{F_i^{Init} \in \mathbb{R}^{H_i \times W_i \times d_0} | i = 1, \dots, N\}$ on every reference view, where H_i and W_i are the size of the i -th reference image and d_0 is its dimension. Then, on every reference view, we concatenate its F_i^{Init} with its image C_i and feed them into a CNN

M to produce a feature map $F_i \in \mathbb{R}^{H_i \times W_i \times d}$ by

$$F_i = M(F_i^{Init}, C_i). \quad (11)$$

Since the reference ray that intersects a specific sample point may not be emitted from a pixel of integer coordinates, the feature vector f of this reference ray is computed by bilinear interpolation of the feature vectors at its four closest pixels on the feature map F_i on the corresponding reference view.

An alternative way is to directly interpolate the parameter feature map F_i^{Init} to get the feature vector f . However, such an alternative way produces worse rendering results. The reason is that the CNN applied in Eq. 11 associates nearby reference rays by convolutions and nearby reference rays usually have similar visibility probability distributions. Furthermore, the CNN M enables the feature vectors f to encode the local image patterns on the image C , which provide more information about reference rays.

3.4 Scene-specific training with DR

Given N reference views with known poses of a specific scene, we can construct a NeuRay representation on these reference views from scratch and render images of novel views in the scene.

Trainable parameters. In the scene-specific training for the direct rendering, the trainable parameters of NeuRay includes: (1) the feature vectors $\{F_i^{Init}\}$ defined on every reference view; (2) the parameters of the MLP network ϕ used in Eq. 4; (3) the weights of the CNN M used in Eq. 11; and (4) the optional regularization term Λ in Eq. 9 if we make it trainable.

Loss. To train NeuRay on the given reference views, we randomly select a reference view as a query view and use other neighboring reference views as the working views to render the selected query view, which results in a rendering loss

$$\ell_{DR} = \sum \|c_{o,DR} - c_{gt}\|^2, \quad (12)$$

where $c_{o,DR}$ is the output color in Eq. 1, c_{gt} is the ground-truth color.

Discussion. The scene-specific training of DR relies on the pixel color consistency of neighboring working views to learn the visibility probabilities on reference rays. In order to minimize the ℓ_{DR} , the optimization process will adjust probabilities on reference rays to increase the hitting probabilities of sample points that have the same color as the ground truth pixel color. Meanwhile, due to the regularization term in spherical harmonics fitting, these sample points must satisfy that the colors of their reference rays are consistent so that a smooth harmonic color function can be fitted. Although direct rendering already produces satisfactory rendering results, it only utilizes color consistency but not make use of the important clues from local feature matching for finding surfaces. Hence, to further improve the rendering quality, we will next consider applying neural networks to incorporate local feature matching in NeuRay. The resulted rendering mode is called *Network Rendering*.

3.5 Network rendering

In Network Rendering (NR), we still sample the query ray and use the Eq. 1 and Eq. 2 to compute the color c_o for the pixel. The difference of NR from DR is that the alpha value e_i and the color c_i are computed by a deep rendering network. The rendering network follows a similar network design as IBRNet [Wang et al. 2021b],

which predicts blending weights to compute colors and applies transformer on the query ray to compute alpha values.

Computation of e_i . For the j -th reference ray of point p_i , we first extract its high-level descriptor by fusing its properties, including the feature vector $f_{i,j}$ associated on reference rays, the visibility $v_{i,j}$, the hitting probability $h_{i,j}$, the color $c_{i,j}$, and the view direction difference $\Delta r_{i,j} = r - r_{i,j}$ between the query view and the reference view. The fusion is implemented by

$$f'_{i,j} = G(f_{i,j}, v_{i,j}, h_{i,j}, c_{i,j}, \Delta r_{i,j}), \quad (13)$$

where G is an MLP network. Since the $f_{i,j}$ is produced by a CNN, it already encodes local image features inside.

After having the multi-view descriptors $\{f'_{i,j} | j = 1, \dots, N\}$ from reference rays of different input views, we further aggregate them into a feature vector f_i on the point p_i using an aggregation network which is in charge of checking the feature consistency among input views. Next, we apply a self-attention transformer on all features $\{f_i | i = 1, \dots, K\}$ along the query ray to get refined features $\{f'_i\}$. Finally, we decode the density value d_i from the refined feature f'_i by another MLP network, which is used in the computation of the alpha value e_i by $e_i = 1 - \exp(-\max(d_i, 0) * l_i)$.

Computation of c_i . We apply an MLP network on the concatenation of $f'_{i,j}$ in Eq. 13 and the view direction difference Δr_j to produce blending weights $w_{i,j}$ and compute the color for this point by $c_i = \sum_j w_{i,j} c_{i,j}$. Details of all network architectures can be found in the supplementary materials.

Discussion about DR and NR. NR of NeuRay adopts a rendering network to exploit the local feature consistency. Such a rendering network is more flexible than the direct combination of probability distributions in the DR. In general, the NR of NeuRay produces better rendering results than the DR. However, the DR of NeuRay is also useful due to its efficiency. As shown in experiments in Sec. 4.3, we can first run DR once to discard sample points with small hitting probabilities and then only feed retained points to NR, which speeds up the rendering while keeps the rendering quality.

3.6 Scene-specific training with DR-NR combination

In this section, we will introduce how to train a NeuRay representation with DR and NR simultaneously. Besides the trainable parameters in DR, NR additionally introduces trainable parameters in rendering networks. In addition to the rendering loss of DR, we also apply the following two losses.

Losses. First, similar to DR, we can also construct an NR rendering loss for training, which is

$$\ell_{NR} = \sum \|c_{o,NR} - c_{gt}\|^2, \quad (14)$$

where $c_{o,NR}$ is the output color from NR. Second, since every query ray itself is also a reference ray during training, we can also compute the hitting probability $h(z_i, z_{i+1})$ directly using Eq. 6. To enforce the consistency between the $h(z_i, z_{i+1})$ and the $h_{i,NR}$ predicted by the network rendering, we apply a cross entropy (CE) loss between them by

$$\ell_{consist} = \frac{1}{K} \sum_{i=1}^K CE(h_{i,NR}, h(z_i, z_{i+1})). \quad (15)$$

The training loss is the sum of all losses, that is

$$\ell = \ell_{DR} + \ell_{NR} + \ell_{consist}. \quad (16)$$

Memory interpretation. Another way to interpret the scene-specific training process of NeuRay with NR and DR is that NeuRay adds a memory for the recent stereo matching based rendering methods [Chibane et al. 2021; Wang et al. 2021b; Yu et al. 2020] to memorize predicted surfaces. We illustrate this by the example shown in Fig. 5.

- (1) In Fig. 5 (a), the rendering network in NR checks the local feature consistency of neighboring views B, C and D to predict $h_{i,NR}$ on the query ray of the query view A which is also a reference view actually.
- (2) Applying the consistency loss $\ell_{consist}$ forces the trainable parameter F^{Init} on the view A to produce parameters $[\alpha, \mu, \sigma]$ that form a hitting probability $h(z_i, z_{i+1})$ consistent with the predicted $h_{i,NR}$. This actually can be interpreted that the predicted $h_{i,NR}$ is memorized in the trainable parameter F^{Init} . Memorizing the $h(z_i, z_{i+1})$ also means the memorization of an equivalent visibility probability, because both visibility and hitting probability are computed from $[\alpha, \mu, \sigma]$.
- (3) When using view A, B and C to render the view D, NeuRay is able to know that green point in Fig. 5 (b) is invisible to view A by checking the memorized visibility on A. Thus, NeuRay overcomes the view inconsistency caused by occlusion and predicts an accurate h for view D.
- (4) The predicted h on view D is also memorized as stated Step (2) and helps the h -prediction of other views.

Such a memorization mechanism enables the network to refine the NeuRay representation better and better during training. In contrast, existing methods [Chibane et al. 2021; Wang et al. 2021b; Yu et al. 2020] lack such memorization mechanism thus are unable to handle complex occlusions in the scene.

3.7 Generalization and finetuning

NeuRay can not only be trained on a specific scene but also be generalized across scenes. This is achieved by constructing a feature initialization network to generate initialized F^{Init} and the feature initialization network along with all other networks used in NeuRay can be shared across different scenes, as shown in Fig. 6. After being trained on a set of scenes, these networks can be directly applied to unseen scenes for view synthesis.

Feature initialization network. For each input view, we first use a multi-view stereo (MVS) algorithm, e.g. Colmap [Schönberger et al. 2016], to estimate its depth map. Then, the estimated depth and the image are concatenated and fed into a series of convolutional layers to produce the initialized F^{Init} . The rationale of using depth is that F^{Init} encodes the visibility probabilities of rays which are highly related to the depth of this view. The detailed architecture of all networks is described in the supplementary material.

Generalization training. The feature initialization network, the MLP network ϕ , the CNN M and the rendering network are trained on a set of training scenes. In each training step, we randomly select a scene and sample N reference views and a query view in the selected scene. Then, the depths and images of reference views

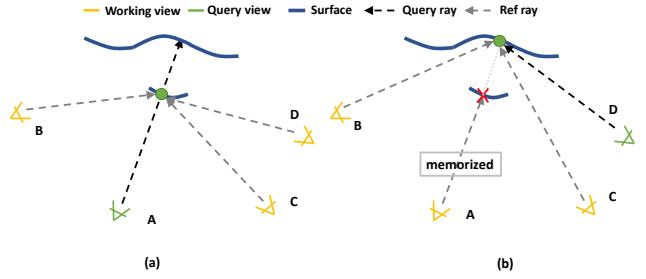


Fig. 5. Illustration for the memory interpretation. All views are input reference views. (a) During training, when rendering view A using view B, C, and D, the predicted hitting probabilities and the corresponding visibility probabilities will be memorized on view A. (b) The memorized visibility on view A will be used in occlusion inference when rendering view D.

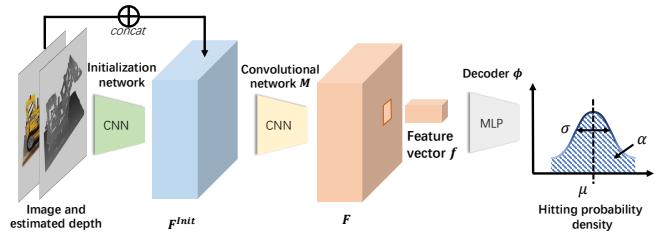


Fig. 6. The generalization NeuRay construction takes an image and its estimated depth as inputs, processes them by two CNNs, and outputs a feature map F which can be interpolated to get the feature vector f of a reference ray. The feature vector can be decoded by an MLP ϕ to the distribution parameters μ, σ and α used in Eq. 3. All these networks are shared across different scenes.

are encoded by the feature initialization network to feature maps $\{F_i^{Init}\}$. Finally, the feature maps $\{F_i^{Init}\}, \phi, M$ and the rendering network are used in DR (Sec. 3.3) and NR (Sec. 3.5) to render the image of the query view. Both rendering losses ℓ_{DR} and ℓ_{NR} are used in the training. Besides, we also include a depth loss in the generalization training, as introduced in Appendix A.5.

Finetuning on a specific scene. Given all the networks pre-trained on the training scenes, we first apply the initialization network to produce a set of initialized $\{F_i^{Init}\}$ on every reference view on this scene. Then, the feature initialization network is discarded but its predicted $\{F_i^{Init}\}$ are treated as a set of trainable parameters, which is trained along with the subsequent networks (ϕ, M and the rendering network) on this scene. We adopt the same losses and training procedure as used in the scene-specific training in Sec. 3.6.

4 EXPERIMENT

4.1 Experimental Protocols

Models and baselines. To validate the effectiveness of NeuRay representation, we evaluate the NeuRay with both DR and NR. We conduct experiments in the following three settings: (1) *scene-specific setting*: we optimize the representation from scratch on a specific scene; (2) *generalization setting*: NeuRay is trained on training scenes

Table 1. PSNR on the NeRF Synthetic dataset.

Type	Method	Steps	Lego	Chair	Drums	Ficus	Hotdog	Materials	Mic	Ship	Avg.
Scene-specific	NeRF	200k	31.39	33.59	26.00	29.97	37.75	30.52	33.08	31.11	31.68
	NeuRay-DR	200k	32.07	34.59	26.39	29.05	37.49	31.64	37.25	31.44	32.49
	NeuRay-NR	200k	33.79	36.37	27.50	31.06	39.57	32.73	38.12	33.14	34.04
Generalization	PixelNeRF	0	22.46	24.16	20.56	24.82	25.10	22.43	29.47	24.27	24.15
	IBRNet	0	26.77	28.67	23.50	25.48	33.20	26.32	33.61	28.60	28.27
	NeuRay-DR	0	28.60	29.93	23.56	24.11	34.18	25.81	34.13	28.48	28.60
	NeuRay-NR	0	29.77	31.90	24.96	25.60	34.72	28.15	35.38	29.24	29.97
Finetune	NeRF	10k	25.11	27.24	22.61	25.28	30.96	25.37	28.78	26.26	26.45
	IBRNet	10k	28.83	31.97	24.53	27.78	36.09	30.54	35.14	29.85	30.59
	NeuRay-DR	10k	30.81	31.97	25.50	25.88	34.40	28.66	36.09	29.86	30.40
	NeuRay-NR	10k	31.38	32.48	26.17	28.33	37.04	30.05	36.67	30.20	31.54
	IBRNet	100k	28.96	32.00	25.06	28.13	36.83	30.90	36.23	29.80	30.99
	NeuRay-DR	100k	31.77	33.13	26.37	27.63	37.20	31.45	37.16	31.61	32.04
	NeuRay-NR	100k	33.46	35.71	27.29	30.43	39.48	32.30	38.04	33.02	33.72

Table 2. PSNR on the DTU dataset.

Type	Method	Steps	Birds	Tools	Bricks	Snowman	Avg.
Generalization	PixelNeRF	0	21.83	16.06	15.99	20.51	18.60
	IBRNet	0	29.22	22.98	23.62	28.35	26.04
	NeuRay-DR	0	29.70	22.16	23.84	26.64	25.59
	NeuRay-NR	0	31.15	24.44	27.18	28.45	27.81
Finetune	IBRNet	10k	31.97	25.77	28.35	29.45	28.89
	NeuRay-DR	10k	31.10	24.87	26.19	28.84	27.75
	NeuRay-NR	10k	33.03	25.94	28.66	29.47	29.23
	IBRNet	100k	32.80	24.83	28.78	29.44	28.96
	NeuRay-DR	100k	32.22	25.56	27.92	29.06	28.69
	NeuRay-NR	100k	33.98	26.60	29.65	29.58	29.95

Table 3. PSNR on the forward-facing dataset. "Gen" means the generalization setting. "Ft" means the finetuning setting.

Setting	PixelNeRF	LLFF	IBRNet	NeuRay-DR	NeuRay-NR
Gen	20.15	25.74	26.08	24.27	26.15
Ft-10k	21.47	N/A	26.69	25.56	27.15
Ft-100k	21.95	N/A	26.71	26.03	28.03

and tested on unseen scenes. Note we only train one generalization model and apply it on all test datasets; (3) *finetuning setting*: we finetune the generalization model in with the training images of the target testing scene. In the scene-specific setting, we use NeRF [Mildenhall et al. 2020] as our baseline method. In the generalization and finetuning setting, we compare with IBRNet [Wang et al. 2021b] and the PixelNeRF [Yu et al. 2020]. All the baseline models are trained with the same training data and training strategy. The quantitative results are reported in Peak Signal-to-Noise Ratio (PSNR) as our main metrics.

4.1.1 Datasets. We use two kinds of evaluation datasets, the object dataset and the forward-facing dataset.

Object dataset. The object dataset includes the NeRF synthetic dataset [Niemeyer et al. 2020] and the DTU dataset [Jensen et al. 2014], both of which contain objects with complex structures and have severe self-occlusions. The NeRF synthetic dataset has 8 objects, each of which contains 100 images as reference views and the other 200 test images as query views. For the DTU dataset, we select 4 scenes as testing scenes and the rest scenes are used in the training of the generalization model. For each scene, we rule out some bad views which contain inconsistent light conditions as done in PixelNeRF [Yu et al. 2020]. Among the remaining 34 views, we apply farthest point sampling on the camera locations to select 17 views as reference views and use the other 17 views as query views.

Forward-facing dataset. The forward-facing dataset is the LLFF dataset [Mildenhall et al. 2019], which contains 8 scenes with clutter objects and complex background. Each scene contains 20 to 62 images. We follow the same train-test images split for each scene as previous methods [Mildenhall et al. 2020; Wang et al. 2021b], which uses 1/8th of the images as test sets. The evaluation resolution is

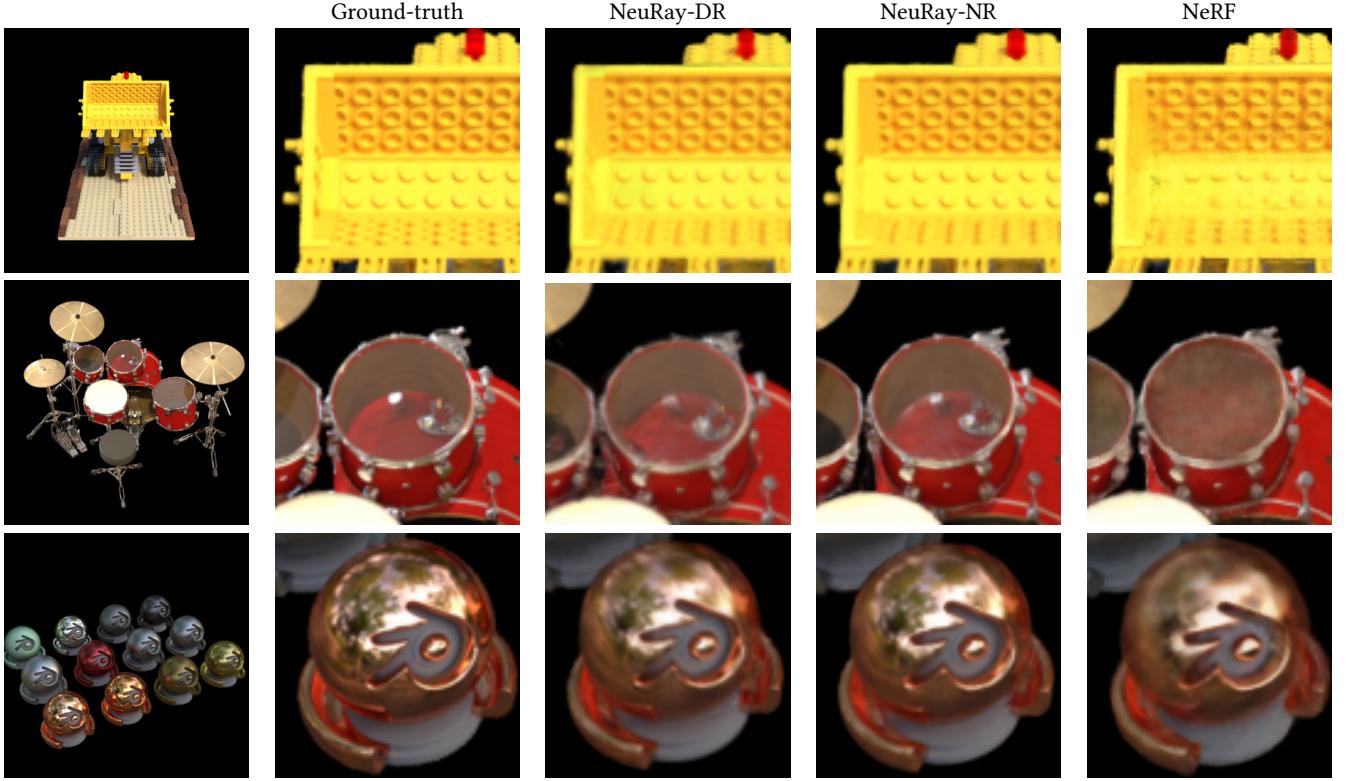


Fig. 7. Qualitative results in the scene-specific training with 200k training steps. NeuRay can recover the subtle regions, like the texture on the Lego, the transparent drum surface or the reflective materials, more clearly than NeRF with the same training steps.

400×300 for both the DTU dataset and the forward-facing dataset. The resolution is 400×400 for the NeRF synthetic dataset and all the test images in two object datasets use black backgrounds.

Training dataset. In order to train the generalization model, we use the following three datasets: (1) synthetic Google Scanned Object dataset [Google 2021], which contains 1023 objects with 250 rendered images on each object; (2) the forward-facing training datasets from IBRNet [Wang et al. 2021b] and (3) the rest training scenes of the DTU dataset.

4.1.2 Implementation details. To render a query view, NeuRay uses its 8 neighboring reference views as working views. The number of sample points K on a query ray is 128. To train NeuRay, we use Adam [Kingma and Ba 2014] as the optimization method. Training NeuRay for 10k steps and 200k steps costs about 40 minutes and 12 hours respectively on a single 2080 Ti GPU. We use the degree 0-3 of spherical harmonic functions in the color fitting of DR. The regularization terms are made to be trainable in the scene-specific setting but are fixed at [0,0.001,0.005,0.05] for degree 0 to 3 in the generalization setting and the finetuning setting. Since the initialization of a NeuRay representation requires estimated depth maps, we use the Colmap [Schönberger et al. 2016] to estimate depth on all datasets. More details can be found in the supplementary materials.

4.2 Comparison with baselines

The quantitative results on the NeRF synthetic dataset, the DTU dataset and the forward-facing LLFF dataset are shown in Table 1, Table 2 and Table 3, respectively. The qualitative results in three settings on the NeRF synthetic dataset are shown in Fig. 7, Fig. 8 and Fig. 9, respectively. More qualitative results on the DTU dataset and forward-facing LLFF dataset and videos of rendering results can be found in the supplementary materials.

4.2.1 Scene-specific setting. The results in the scene-specific setting of Table 1 show that NeuRay can achieve a performance that is comparable or even better than NeRF [Mildenhall et al. 2020]. From Fig. 7, we can see that NeRF needs more optimization steps to recover subtle regions like the texture on the Lego, the transparent drum surface or the reflective materials. In comparison, both DR and NR of NeuRay can render these regions more clearly with the same training steps. Comparing NR with DR, we can find that they both correctly reveal the geometry of the scene but the renderings of NR are more clear than those of DR. The reason is that NR uses a rendering network, which is more flexible than the explicit combination of probabilities on reference rays in DR.

4.2.2 Generalization setting. In the generalization setting, NeuRay-DR already achieves comparable results with IBRNet on the NeRF synthetic dataset and the DTU dataset but worse results on the

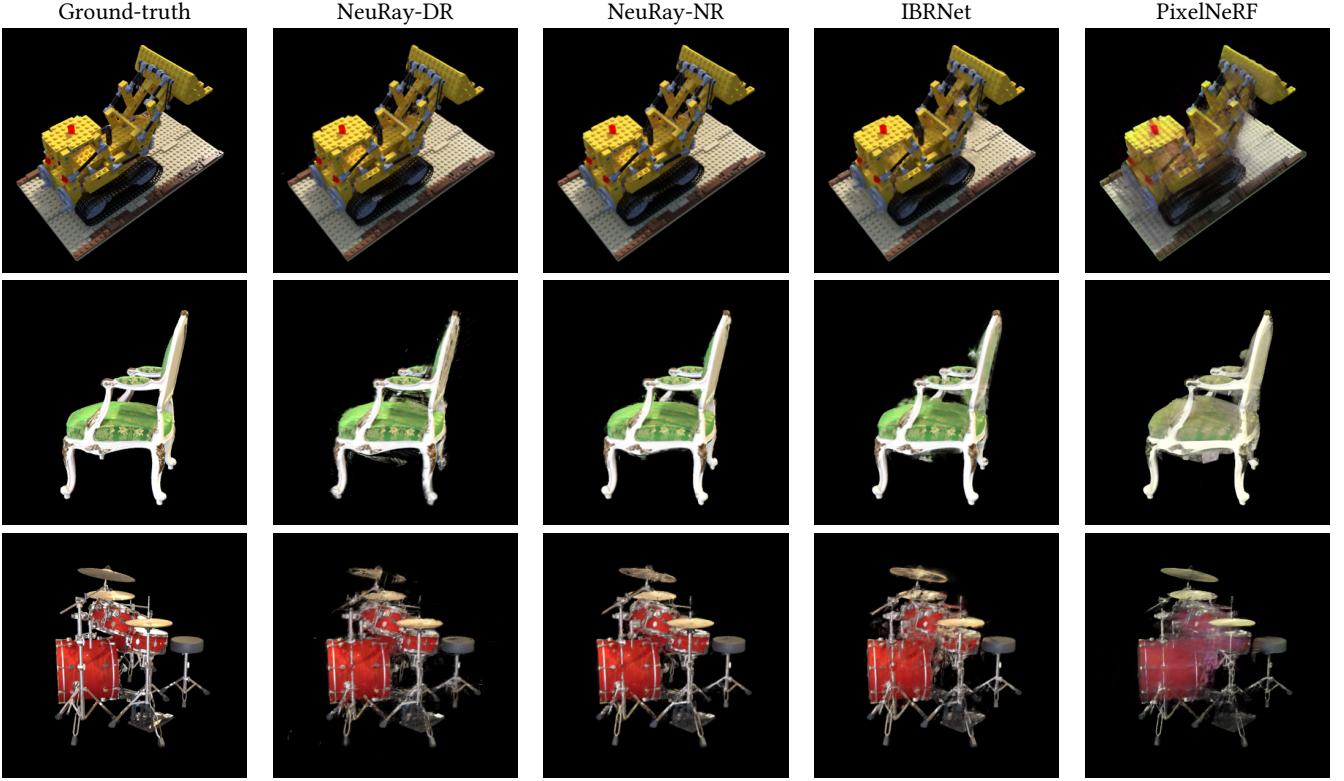


Fig. 8. Qualitative results in the generalization setting. NeuRay performs better in regions with severe self-occlusion, e.g. the plane occluded by arms of Lego model, the chair plane occluded by the arm of the chair and the center region of drums.

forward-facing dataset than LLFF and IBRNet. Meanwhile, our NeuRay-NR outperforms all baselines on all datasets in this setting. As shown in Fig. 8, in comparison with baselines, NeuRay improves the rendering quality in regions with severe self-occlusion (e.g. the bottom plane occluded by arms of Lego model, the chair plane occluded by the arm of the chair and the center region of drums). The reason is that, by taking the estimated depth maps from Colmap, NeuRay is able to take the occlusion relationship between rays into account while baselines are unable to do this. Since the input estimated depth maps are not accurate on all input views and NeuRay-DR does not involve any trainable parameters in the rendering process to correct the erroneous input depth maps, NeuRay-DR does not achieve better performance in all cases. Meanwhile, NeuRay-NR has a rendering network to additionally consider the local feature consistency thus is able to select correct depth maps for the occlusion inference.

4.2.3 Finetuning setting. The results in the finetuning setting show that the proposed NeuRay-NR outperforms all other baselines with the same training steps. Table 1 shows that NeuRay-NR with 10k finetuning steps already achieves comparable performance (PSNR 31.54) with the NeRF training with 200k steps (PSNR 31.68). From Fig. 9, we find that training NeRF from scratch for 10k steps is not enough for convergence, which results in blurred images. In comparison, both the pretraining of NeuRay and IBRNet reduce the

required training steps for satisfactory rendering quality. However, further finetuning with 100k steps performance only brings slight improvements (+0.59 PSNR) to IBRNet but significant improvements (+2.2 PSNR) to NeuRay, as shown in Table 1. The reason is that IBRNet lacks the ability to memorize the predicted surfaces as stated in Sec. 3.6. In contrast, NeuRay memorizes the surface locations on feature vectors of reference rays, which enables further finetuning to refine them for better rendering results. On the DTU dataset and the forward-facing LLFF dataset, we can observe similar results.

4.3 Speed-up rendering process

The rendering of NeuRay can be accelerated from two aspects. First, since NeuRay represents a scene by reference rays on reference views, the features maps on these reference views can be computed beforehand once and used in all subsequent rendering processes, which is called "feats cache". Moreover, instead of interpolating on feature maps and applying decoders to compute the distribution parameters $[\mu, \sigma, \alpha]$, we can compute these distribution parameters beforehand on reference views and directly interpolate on these distribution parameters, which is called "prob cache". Second, since the computation of a hitting probability h_i on a query ray with DR is a simple weighted average, we can first use DR to compute the hitting probabilities of sample points and discard the points with small hitting probabilities. Only retained points will be used in the

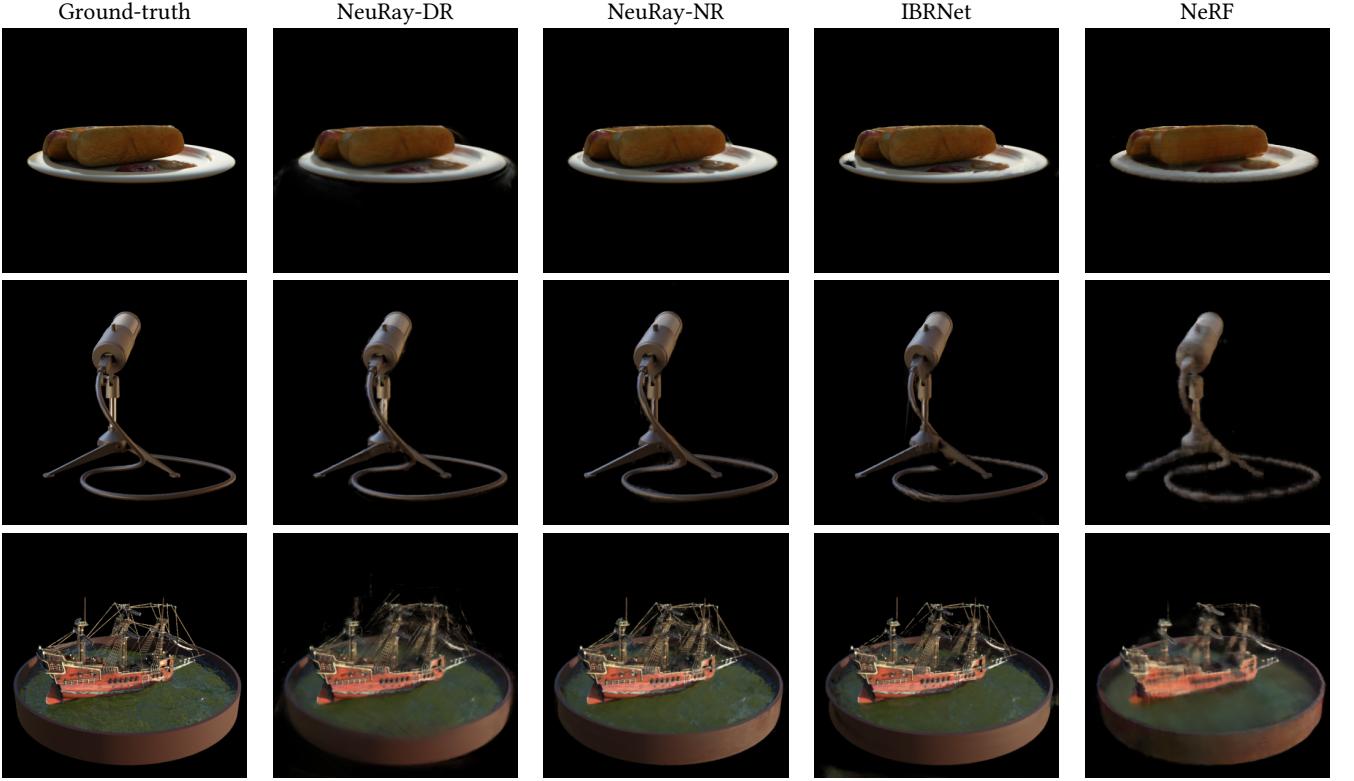


Fig. 9. Results in the finetuning setting. All models are trained on the scene for 10k steps. NeRF is trained from scratch while NeuRay and IBRNet are first initialized by their generalization models.

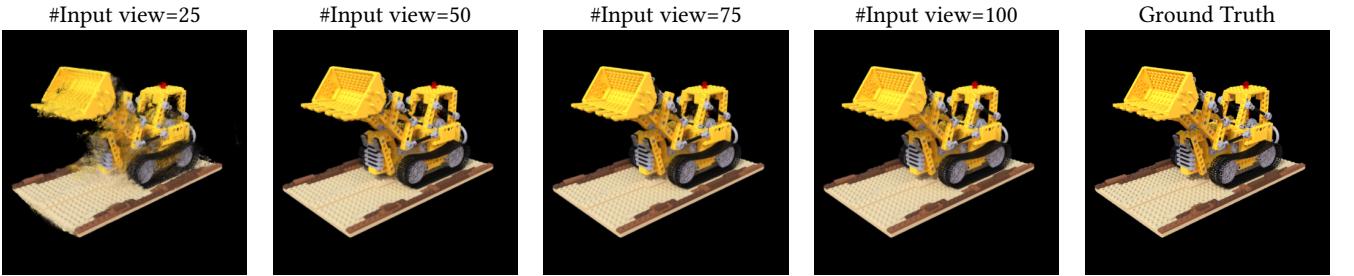


Fig. 10. Qualitative results of NeuRay-NR in the generalization setting using different input view numbers. With only 25 input views around the scene, obvious artifacts appear since the supporting neighboring view is too far from the query view. When there are 50 input views, NeuRay is already able to produce rendering images of satisfactory quality.

color prediction of DR or the rendering network of NR, which will slightly degenerate the rendering quality but speed up the rendering process significantly.

In Table 4, we report the rendering time and quality of NeuRay on the Lego of the NeRF synthetic dataset with or without two speeding-up strategies. In the table, the first rows of NeuRay-DR and NeuRay-NR are the original models which cost 6.14 seconds and 11.41 seconds respectively to render a 400×400 image. The model of the last row of NeuRay-DR in the table uses the "prob cache" and discards sample points with hitting probabilities less than 0.01,

which only takes 0.89 seconds to render an image. The model of the last row of NeuRay-NR only processes 8 sample points with largest hitting probabilities on a query ray, which costs about 1.12 seconds on a query view. After applying these two speeding-up strategies, the most time-consuming part is the projection of all 128 sample points onto 8 neighboring working views for the hitting probability computation in DR, which costs about 0.6 seconds for one image. This part can be further sped-up by constructing a coarse shape from NeuRay-DR in the 3D space and only sampling few points near the surface for rendering, which we will study in future works.

Table 4. Rendering time and quality on the Lego of the NeRF synthetic dataset using NeuRay trained with the scene-specific setting. "Cache" means what is cached for rendering. "feats" means the model caches feature map F and interpolate on F for rendering. "prob" means the model caches probability distribution parameters $[\mu, \sigma, \alpha]$ and directly interpolate on these parameters. "Thresh" is the threshold on hitting probabilities to discard sample points. "Top k " means the model only uses k sample points with largest hitting probabilities on every query ray. "Time" is the time cost to render a 400×400 image on a 2080 Ti GPU.

Model	Cache	Thresh	Top k	Time(s)	PSNR
NeuRay-DR	feats	0	N/A	6.14	32.07
NeuRay-DR	prob	0	N/A	3.93	32.03
NeuRay-DR	prob	0.001	N/A	1.05	32.00
NeuRay-DR	prob	0.01	N/A	0.89	31.99
NeuRay-NR	feats	N/A	128	11.41	33.79
NeuRay-NR	feats	N/A	8	4.13	33.61
NeuRay-NR	prob	N/A	8	1.12	33.13

Table 5. Ablation studies on the Drums model in the NeRF synthetic dataset. Results are reported in PSNR and achieved in the finetuning setting with 100k steps. "Full model" uses all losses and models. "Without $\ell_{consist}$ " does not use the consistency loss between two kinds of hitting probabilities. "Without ℓ_{DR} " does not apply DR on NeuRay for finetuning. "Without NR" does not use NR. "Average colors" means that the model does not use spherical harmonics for color fitting but simply average colors of the working reference views instead.

Description	NeuRay-DR	NeuRay-NR
Full model	26.37	27.29
Without $\ell_{consist}$	25.64	26.90
Without ℓ_{DR}	N/A	27.28
Without ℓ_{NR}	23.66	N/A
Average colors for DR	25.31	27.12

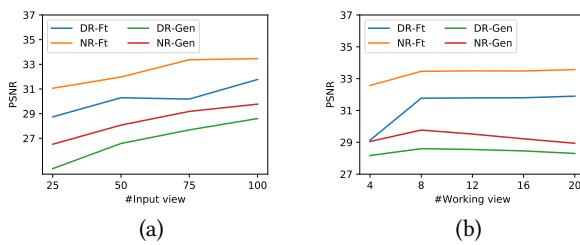


Fig. 11. PSNR of NeuRay on the Lego using different numbers of input view (a) or different numbers of working view (b). "Gen" means the generalization setting while "Ft" means the finetuning setting with 100k steps.

4.4 Ablation studies

We conduct ablation studies on the Drums from the NeRF synthetic dataset in the finetuning setting. As shown in Table 5, the $\ell_{consist}$ benefits the rendering quality of both DR and NR of NeuRay. The model without the ℓ_{DR} still has the $\ell_{consist}$ to guarantee that the

Table 6. PSNRs on the NeRF-synthetic dataset with resolution of 800×800 and the forward-facing LLFF dataset with resolution of 1008×756.

Setting	Method	Dataset	
		NeRF-Synthetic	Forward-Facing
Generalization	MVSNeRF	22.67	17.56
	LLFF	24.88	24.13
	IBRNet	25.49	25.13
	NeuRay-NR	27.24	25.15
Finetune	MVSNeRF	27.21	26.25
	IBRNet	28.14	26.73
	NeRF	31.01	26.50
	NeuRay-NR	32.42	26.78

Table 7. PSNR on Fern/Horns from the forward-facing LLFF dataset and Drums/Ficus from the NeRF synthetic dataset. "single" means a single logistics distribution while "mixture" means a mixture of two logistics distributions. Models are trained from scratch on each scene with 200k steps.

Choice of S	Type	Fern	Horns	Drums	Ficus
Single	NR	28.04	30.12	27.50	31.06
	DR	25.47	26.12	26.39	29.05
Mixture	NR	28.32	30.24	27.89	31.34
	DR	27.25	27.17	27.05	30.10

feature vectors on the reference rays are able to encode the hitting probabilities, so it achieves similar performance in NR as the full model. The model without ℓ_{NR} does not have a rendering network for local feature matching and thus is much worse than the full model. The final model does not apply spherical harmonics functions for color fitting but simply uses the weighted average colors of input views with hitting probabilities as weights, which also produces worse results in DR.

4.5 Analysis

4.5.1 Number of input views. To investigate how the performance degrades as the decrease of the number of input views, we reduce the input views of the Lego of the NeRF synthetic dataset from 100 to 75, 50 and 25 by the farthest point sampling on camera locations. The PSNR plots are shown in the Fig. 11 (a) and qualitative results are shown in Fig. 10. As expected, the performance degrades reasonably as the reference views become sparser. The qualitative results show that artifacts may appear when there are only 25 input views, which are too sparse to capture the detailed shape of the scene. However, increasing the input view number to 50 already suffices for our method to produce images of satisfactory quality.

4.5.2 Number of Working view. To show how the number of working view affects the performance, we conduct an experiment on the Lego using different numbers of working views. The results are shown in Fig. 11 (b). Both NR and DR in the generalization setting perform best with 8 working views, because they are trained with 8 working views. Meanwhile, after finetuning, the performance of both NR and DR improves slightly with more working views

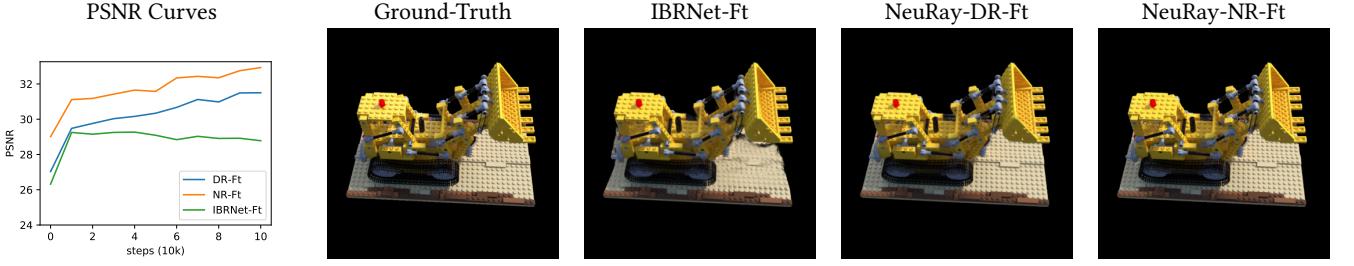


Fig. 12. Curves of PSNR on different steps and qualitative results on the 100k steps. Note the PSNR is computed on a validation set with 6 images.

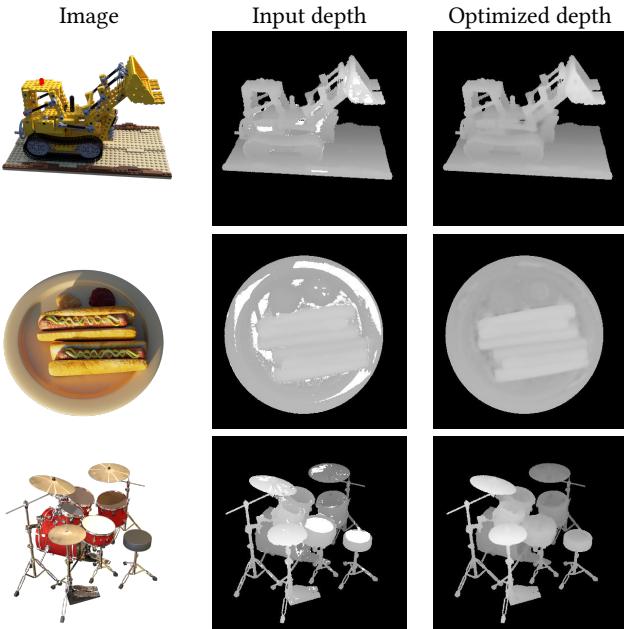


Fig. 13. Input depth maps estimated by Colmap [Schönberger et al. 2016] and optimized depth maps by NeuRay on the same reference views. The optimized depth values are the means μ of all reference rays. In the input depth map, there are white regions which are the failed regions for the external MVS algorithm. Optimizing NeuRay representation can find correct depth values for these failed regions. Note backgrounds are masked out for clear visualization.

because finetuning refines the NeuRay representation and brings about more consistency between views of larger distances.

4.5.3 Optimization progress. To show how the performance of different models improves in the optimization process, we finetune different models on the Lego and draw the plots of PSNR on different training steps on a small validation set in Fig. 12. The plots show that IBRNet only achieves improvement on the first 10k steps and its performance even degrades slightly on the validation set with more training steps due to overfitting. In contrast, NeuRay consistently improves the performance with more finetuning steps. The reason is that IBRNet lacks a memorization mechanism as explained in the memory interpretation in Sec. 3.6.

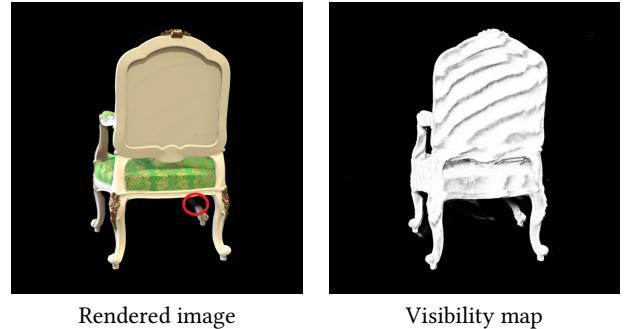


Fig. 14. The red circle contains a region that is invisible to all neighboring working views. Thus, NeuRay is unable to correctly render this region. However, we can compute a visibility map of the query view from NeuRay to know which regions are not visible by working views. (The striped texture on the visibility map is caused by the sampling interval of the uniform sampling on query rays.)

4.5.4 Depth fusion. By taking the finetuned means μ of reference rays as the depth values on this reference view, we can get a depth map that is complete and consistent with other views. In Fig. 13, we show some examples of the input incomplete depth maps and the output refined depth maps, which demonstrates that optimizing NeuRay representation can enforce the consistency between views and rectify erroneous depth values. In light of this, NeuRay may also be used as a depth fusion method or a consistency check for MVS algorithm, which takes coarse depth maps as input and outputs refined consistent depth maps. However, unlike commonly-used consistency check which simply discards erroneous depth values, optimizing NeuRay is able to correct these erroneous depth values.

4.5.5 Higher resolution rendering. To show the ability of NeuRay to render higher resolution images, we further provide results on the forward-facing LLFF dataset with the resolution of 1008×756 and the NeRF synthetic dataset with the resolution of 800 × 800 in Table 6. In the table, we also include the performance of recent MVSNeRF [Chen et al. 2021] reported in their paper. Note the performance of NeuRay-NR in the generalization setting already outperforms that of MVSNeRF in the finetuning setting on the NeRF synthetic dataset.

4.5.6 Mixture of logistics distributions. By default, NeuRay uses a logistics distribution as the hitting probability distribution of a

reference ray. Since a logistics distribution has only one peak, it has difficulty in representing rays that hit semi-transparent objects or rays in a clutter scene, which require multiple peaks on the hitting probabilities along the ray direction. Though, by combination of probabilities on reference rays, we can still construct a multiple-peak distribution on a query ray to render a semi-transparent object, e.g. the drums in Fig. 7, optimizing NeuRay with the consistency loss $\ell_{consist}$ in Eq. 15 is not accurate in this case, which makes the model struggle to improve during finetuning. To remedy this, we can use a mixture of multiple logistics distributions to get better performance, as shown in Table 7. Replacing the logistics distribution with a mixture of two logistics distributions brings 1-2 PSNR on the results of DR and slight improvement on the results of NR.

4.5.7 Limitations. As an image-based rendering method, NeuRay needs to select a set of working views to render a novel view image. Hence, NeuRay is unable to render pixels that are invisible to all working views, though these pixels may be visible in other reference views. We show an example in the Fig. 14, in which the region in the red circle is invisible to all working views. However, since the visibility probabilities of reference rays are encoded in the NeuRay representation, we can decode a visibility map of the query view from NeuRay by accumulating the visibility along the query ray. Such visibility map will show the invisible regions.

5 CONCLUSION

In this paper, we proposed a novel neural representation, called Neural Ray (NeuRay), for the novel view synthesis task. NeuRay regards every input reference view as a bunch of reference rays and associates a feature vector with every reference ray to encode the visibility of this ray. Then, an unseen query view can be directly rendered from the proposed NeuRay representation or combined with stereo matching rendering networks to get photo-realistic rendering results. Experiments on the DTU dataset, the NeRF synthetic dataset and the forward-facing LLFF dataset demonstrate that NeuRay can render high-quality images by finetuning on the scene for a few steps or even without any training on the scene.

REFERENCES

- Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. 2020. Neural point-based graphics. In *ECCV*.
- Alexander W. Bergman, Petr Kellnhofer, and Gordon Wetzstein. 2021. Fast Training of Neural Lumigraph Representations. *arXiv preprint arXiv:2106.14942* (2021).
- Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. 2013. Depth synthesis and local warps for plausible image-based navigation. *ACM TOG* (2013).
- Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. 2021. MVSNeRF: Fast Generalizable Radiance Field Reconstruction from Multi-View Stereo. *arXiv preprint arXiv:2103.15595* (2021).
- Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. 2021. Stereo Radiance Fields (SRF): Learning View Synthesis for Sparse Views of Novel Scenes. *arXiv preprint arXiv:2104.06935* (2021).
- Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. 2019. Extreme view synthesis. In *ICCV*.
- Abe Davis, Marc Levoy, and Fredo Durand. 2012. Unstructured light fields. In *Eurographics*.
- Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. 2021. FastNeRF: High-Fidelity Neural Rendering at 200FPS. <https://arxiv.org/abs/2103.10380> (2021).
- Research Google. 2021. Google Scanned Objects. <https://app.ignitionrobotics.org/GoogleResearch/fuel/collections/GoogleScannedObjects>.
- Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. 1996. The lumigraph. In *SIGGRAPH*.
- Marc Habermann, Lingjie Liu, Weipeng Xu, Michael Zollhoefer, Gerard Pons-Moll, and Christian Theobalt. 2021. Real-time Deep Dynamic Characters. *ACM Transactions on Graphics* 40, 4, Article 94 (aug 2021).
- Tong He, John Collomosse, Hailin Jin, and Stefano Soatto. 2020. Geo-PIFu: Geometry and Pixel Aligned Implicit Functions for Single-view Human Reconstruction. *arXiv preprint arXiv:2006.08072* (2020).
- Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. 2018. Deep blending for free-viewpoint image-based rendering. *ACM TOG* (2018).
- Yang Hong, Juyong Zhang, Boyi Jiang, Yudong Guo, Ligang Liu, and Hujun Bao. 2021. StereoPIFu: Depth Aware Clothed Human Digitization via Stereo Vision. *arXiv preprint arXiv:2104.05289* (2021).
- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *CVPR*.
- Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanaes. 2014. Large scale multi-view stereopsis evaluation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 406–413.
- Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. 2016. Learning-based view synthesis for light field cameras. *ACM TOG* (2016).
- Petr Kellnhofer, Lars Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. 2021. Neural Lumigraph Rendering. In *CVPR*.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- Marc Levoy and Pat Hanrahan. 1996. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 31–42.
- Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020a. Neural sparse voxel fields. In *NeurIPS*.
- Lingjie Liu, Marc Habermann, Viktor Rudnev, Kripasindhu Sarkar, Jiatao Gu, and Christian Theobalt. 2021. Neural Actor: Neural Free-view Synthesis of Human Actors with Pose Control. *arXiv:2106.02019 [cs.CV]*
- Lingjie Liu, Weipeng Xu, Marc Habermann, Michael Zollhöfer, Florian Bernard, Hyeongwoo Kim, Wenping Wang, and Christian Theobalt. 2020b. Neural Human Video Rendering by Learning Dynamic Textures and Rendering-to-Video Translation. *TVCG PP* (05 2020), 1–1.
- Lingjie Liu, Weipeng Xu, Michael Zollhoefer, Hyeongwoo Kim, Florian Bernard, Marc Habermann, Wenping Wang, and Christian Theobalt. 2019. Neural rendering and reenactment of human actor videos. *ACM TOG* (2019).
- Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. 2020c. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *CVPR*.
- Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural volumes: Learning dynamic renderable volumes from images. In *SIGGRAPH*.
- Ricardo Martin-Brualla, Noha Radwan, Mehdi Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. 2020. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. <https://arxiv.org/abs/2008.02268> (2020).
- Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.
- Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*.
- Michael Niemeyer and Andreas Geiger. 2020. GIRAFFE: Representing Scenes as Compositional Generative Neural Feature Fields. In *CVPR*.
- Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *CVPR*.
- Sida Peng, Junting Dong, Qianqian Wang, Shangzhan Zhang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. 2021a. Animatable Neural Radiance Fields for Human Body Modeling. *arXiv preprint arXiv:2105.02872* (2021).
- Sida Peng, Yuqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. 2021b. Neural Body: Implicit Neural Representations with Structured Latent Codes for Novel View Synthesis of Dynamic Humans. In *CVPR*.
- Eric Penner and Li Zhang. 2017. Soft 3D reconstruction for view synthesis. *ACM TOG* (2017).
- Amit Raj, Michael Zollhoefer, Tomas Simon, Jason Saragih, Shunsuke Saito, James Hays, and Stephen Lombardi. 2021. PVA: Pixel-aligned Volumetric Avatars. *arXiv preprint arXiv:2101.02697* (2021).
- Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. 2021. KiloneRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. [arXiv:2103.13744 \[cs.CV\]](https://arxiv.org/abs/2103.13744)
- Konstantinos Rematas, Ricardo Martin-Brualla, and Vittorio Ferrari. 2021. ShaRF: Shape-conditioned Radiance Fields from a Single View. <https://arxiv.org/pdf/2102.08860.pdf>

- (2021).
- Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. *NeurIPS* (2015).
- Gernot Riegler and Vladlen Koltun. 2020. Free View Synthesis. In *ECCV*.
- Gernot Riegler and Vladlen Koltun. 2021. Stable View Synthesis. In *CVPR*.
- Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. 2019. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2304–2314.
- Shunsuke Saito, Tomas Simon, Jason Saragih, and Hanbyul Joo. 2020. Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 84–93.
- Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. 2016. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*.
- Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *ECCV*.
- Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. 2019a. Deepvoxels: Learning persistent 3d feature embeddings. In *CVPR*.
- Vincent Sitzmann, Michael Zollhofer, and Gordon Wetzstein. 2019b. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *NeurIPS*.
- Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. 2021. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*.
- Justus Thies, Michael Zollhofer, and Matthias Nießner. 2019. Deferred neural rendering: Image synthesis using neural textures. *ACM TOG* (2019).
- Justus Thies, Michael Zollhofer, Christian Theobalt, Marc Stamminger, and Matthias Nießner. 2020. IGNOR: image-guided neural object rendering. In *ICLR*.
- Alex Trevithick and Bo Yang. 2020. GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering. <https://arxiv.org/abs/2010.04595> (2020).
- Peng Wang, Lingjiu Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021a. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. *arXiv preprint arXiv:2106.10689* (2021).
- Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. 2021b. IBRNet: Learning Multi-View Image-Based Rendering. *arXiv preprint arXiv:2102.13090* (2021).
- Suttisak Wizadwongsak, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. 2021. NeX: Real-time View Synthesis with Neural Basis Expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Minye Wu, Yuehao Wang, Qiang Hu, and Jingyi Yu. 2020. Multi-View Neural Human Rendering. In *CVPR*.
- Zexiang Xu, Sai Bi, Kalyan Sunkavalli, Sunil Hadap, Hao Su, and Ravi Ramamoorthi. 2019. Deep view synthesis from sparse photometric images. *ACM TOG* (2019).
- Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *arXiv*.
- Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. 2020. pixelNeRF: Neural Radiance Fields from One or Few Images. <https://arxiv.org/abs/2012.02190> (2020).

A APPENDIX

A.1 Proof of Property 1

$$\begin{aligned}
 & \prod_{k=1}^{i-1} (1 - e(z_k, z_{k+1})) e(z_i, z_{i+1}) \\
 &= \left(1 - \frac{h(z_1, z_2)}{v(z_1)}\right) \left(1 - \frac{h(z_2, z_3)}{v(z_2)}\right) \dots \frac{h(z_i, z_{i+1})}{v(z_i)} \\
 &= \frac{v(z_1) - h(z_1, z_2)}{v(z_1)} \frac{v(z_2) - h(z_2, z_3)}{v(z_2)} \dots \frac{h(z_i, z_{i+1})}{v(z_i)} \quad (17) \\
 &= \frac{v(z_2)}{v(z_1)} \frac{v(z_3)}{v(z_2)} \dots \frac{h(z_i, z_{i+1})}{v(z_i)} \\
 &= \frac{h(z_i, z_{i+1})}{v(z_1)} \\
 &= h(z_i, z_{i+1})
 \end{aligned}$$

Note that in the third equation, we use $v(z_i) - h(z_i, z_{i+1}) = (1 - t(z_i)) - (t(z_{i+1}) - t(z_i)) = 1 - t(z_{i+1}) = v(z_{i+1})$; In the fifth equation,

we use the $v(z_1) = 1$ because the first depth is assumed to be the near plane which should always be visible. \square

A.2 Motivation of probabilities defined on reference rays

The key idea of NeuRay is to compute the visibility of reference rays so that we can know whether a sample point is visible or not to a particular input view.

A.2.1 From density to visibility. To compute such visibility on a reference ray, an alternative way is to adopt a NeRF-like [Mildenhall et al. 2020] density MLP network for every reference ray. Using the NeRF-like density to compute the visibility can be summarized as follows.

- (1) Sample points p_i on the reference ray for $i = 1, 2, \dots, N_r$. Let us denote the depth for every point as z_i .
- (2) Evaluate the MLP to get the densities d_i on these points.
- (3) Compute the alpha values $e_i = 1 - \exp(-\text{ReLU}(d_i))$.
- (4) Compute the visibility of the point p_i by $v_i = \prod_{j=1}^{i-1} (1 - e_j)$.

However, the evaluation of the density MLP on every sample point on reference rays is too expensive and inefficient. Given 128 sample points on a query ray, 8 working reference views and 32 sample points on every reference ray, we need to evaluate the density MLP $128 \times 8 \times 32$ times for a single query ray, which thus is unacceptable in practice.

A.2.2 From occlusion probability to visibility. NeuRay directly encodes the occlusion probability $t_i = t(z_i)$ so that the visibility can be efficiently computed by $v_i = 1 - t_i$. Obviously, occlusion probability $t(z) \in [0, 1]$ should be a non-decreasing function defined on depth z . Hence, we choose a sigmoid function for t . If such a sigmoid function is regarded as a cumulative density function (CDF) of a logistics probability distribution, such CDF actually corresponds to a probability density function (PDF) of a hitting probability. In other word, NeuRay directly parameterizes the hitting probability of a reference ray as a logistic distribution. A comparison is shown in Fig. 16. NeuRay encodes the occlusion probability for efficient computation of visibility and hitting probability and then derive the corresponding alpha values.

A.3 Detailed cases of DR

To illustrate the alpha values defined in Eq. 7, we provide the curves of a hitting probability and the corresponding alpha values on the same reference ray in Fig. 15 (a).

We discuss detailed three cases in Eq. 8.

- (1) The sample point on the surface. As shown in Fig. 15 (b), both the visibility probabilities and the hitting probabilities of all reference rays are large, which results in a large alpha value on this sample point with Eq. 7.
- (2) The sample point outside the surface. As shown in Fig. 15 (c), the visibility probabilities are large while the hitting probabilities are small, which results in a small alpha value on this sample point.
- (3) The sample point inside the surface. As shown in Fig. 15 (d), both the visibility probabilities and the hitting probabilities of all reference rays are small, which still leads to a large alpha on this sample point.

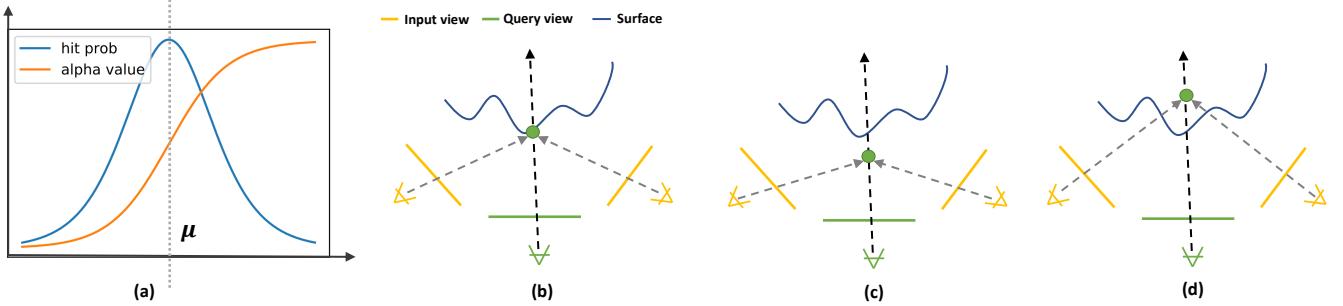


Fig. 15. (a) Alpha values and hitting probabilities on a reference ray. (b) The sample point is on the surface. (c) The sample point is outside of the object. (d) The sample point is inside the object.

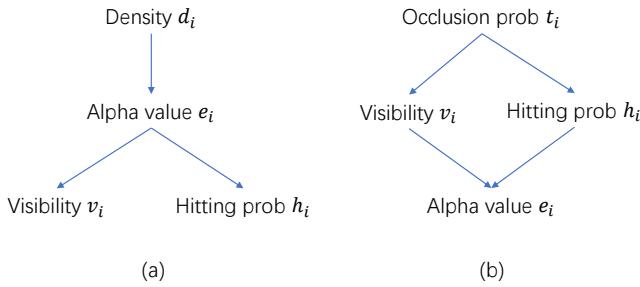


Fig. 16. Comparison between the density in NeRF and the occlusion probability in NeuRay. (a) In NeRF, the network encodes the density to compute the alpha values, the hitting probability and the visibility. (b) In NeuRay, the network directly encodes the parameters of occlusion probability, which enables efficient computation of the visibility and the hitting probability. Then, we derive the corresponding alpha values to the hitting probability, which is Eq. 7.

A.4 Spherical harmonic color fitting

Let us only consider the spherical harmonic fitting for a single point. We omit the subscript i and we have $j = 1, 2, \dots, N$ reference rays on this point. The form of the color function can be written as

$$R(\mathbf{r}; \boldsymbol{\theta}) = \sum_{l=0}^L \sum_{m=-l}^l \theta_l^m Y_l^m(\mathbf{r}), \quad (18)$$

where $\boldsymbol{\theta}$ is all our parameters, $\mathbf{r} \in \mathbb{S}^2$ is the a point on the unit sphere, l is the degree of this spherical harmonic function, L is the max considered degree, m is the order in this degree and Y_l^m is the spherical harmonic function of the degree m and order l . $\boldsymbol{\theta}$ has $(L + 1)^2$ terms for all three color component (RGB). Let us only consider one color component currently and it can be easily extend to three components. The problem 9 can be rewritten as

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \boldsymbol{\theta}^\top (\mathbf{A}^\top \mathbf{W} \mathbf{A} + \Lambda) \boldsymbol{\theta} + \mathbf{b}^\top \mathbf{W} \mathbf{A} \boldsymbol{\theta} + C, \quad (19)$$

where \mathbf{A} has the size of $N \times (L + 1)^2$ contains all terms of Y_l^m , $\mathbf{W} \in \mathbb{R}^{N \times N}$ is a diagonal matrix with elements from the hitting probabilities h_j , Λ is the regularization diagonal matrix, $\mathbf{b} \in \mathbb{R}^{N \times 1}$ contains the color from all reference rays and C is a constant that is

not relevant to $\boldsymbol{\theta}$. Obviously, the solution is given by

$$\boldsymbol{\theta} = \frac{1}{2} (\mathbf{A}^\top \mathbf{W} \mathbf{A} + \Lambda)^{-1} (\mathbf{A}^\top \mathbf{W} \mathbf{b}). \quad (20)$$

A.5 Depth loss

To encourage the output means μ on reference rays to be similar to the input depth for this pixel. We add a depth loss in the generalization training by

$$\ell_{\text{depth}} = \text{SmoothL1}(\|d, \mu\|), \quad (21)$$

where d is the input estimated depth for a reference ray, μ is the predicted mean of the logistics distribution on this reference ray and we apply the smooth L1 loss [Ren et al. 2015] since it is more robust to outliers.