Differentiable Rendering with Reparameterized Volume Sampling

Nikita Morozov
HSE Unversity

Denis Rakitin
HSE University

Oleg Desheulin
HSE University

Dmitry Vetrov*
Constructor University, Bremen

Kirill Struminsky
HSE Unversity

Abstract

In view synthesis, a neural radiance field approximates underlying density and radiance fields based on a sparse set of scene pictures. To generate a pixel of a novel view, it marches a ray through the pixel and computes a weighted sum of radiance emitted from a dense set of ray points. This rendering algorithm is fully differentiable and facilitates gradient-based optimization of the fields. However, in practice, only a tiny opaque portion of the ray contributes most of the radiance to the sum. We propose a simple end-to-end differentiable sampling algorithm based on inverse transform sampling. It generates samples according to the probability distribution induced by the density field and picks non-transparent points on the ray. We utilize the algorithm in two ways. First, we propose a novel rendering approach based on Monte Carlo estimates. This approach allows for evaluating and optimizing a neural radiance field with just a few radiance field calls per ray. Second, we use the sampling algorithm to modify the hierarchical scheme proposed in the original NeRF work. We show that our modification improves reconstruction quality of hierarchical models, at the same time simplifying the training procedure by removing the need for auxiliary proposal network losses.

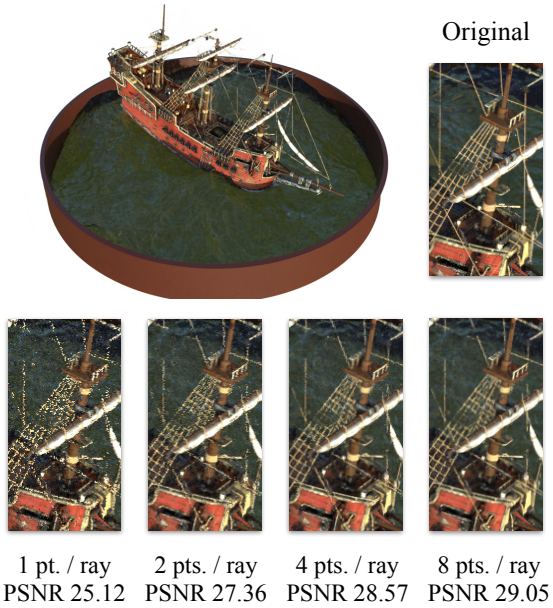

Figure 1: Novel views of a ship generated with the proposed Monte Carlo radiance estimates. For each ray we estimate density and then compute radiance at a few ray points generated using the ray density. As the above images indicate, render quality gradually improves with the number of ray samples, without visible artifacts at eight points per ray.

1 INTRODUCTION

Given a set of scene pictures with corresponding camera positions, novel view synthesis aims to generate pictures of the same scene from new camera positions. Recently, learning-based approaches have led to significant progress in this area. As an early instance, neural radiance fields (NeRF) by Mildenhall et al. (2020) represent a scene via a density field and a radiance (color) field parameterized with a multilayer perceptron (MLP). Using a differentiable volume rendering

*Work done while working at AIRI. Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

algorithm (Max, 1995) with MLP-based fields to produce images, they minimize the discrepancy between the output images and a set of reference images to learn a scene representation.

In particular, NeRF generates an image pixel by casting a ray from a camera through the pixel and aggregating the radiance at each ray point with weights induced by the density field. Each term involves a costly neural network query, and the model has a trade-off between rendering quality and computational load. In this work, we revisit the formula for the aggregated radiance computation and propose a novel approximation based on Monte Carlo methods. We compute our approximation in two stages. In the first stage, we march through the ray to estimate density. In the second stage, we construct a Monte Carlo color approximation using the density to pick points along the ray. The resulting estimate is fully differentiable and can act as a drop-in replacement for the standard rendering algorithm used in NeRF. Fig. 1 illustrates the estimates for a varying number of samples. Compared to the standard rendering algorithm, the second stage of our algorithm avoids redundant radiance queries and can potentially reduce computation during training and inference.

Furthermore, we show that the sampling algorithm used in our Monte Carlo estimate is applicable to the hierarchical sampling scheme in NeRF. Similar to our work, the hierarchical scheme uses inverse transform sampling to pick points along a ray. The corresponding distribution is tuned using an auxiliary training task. In contrast, we derive our algorithm from a different perspective and obtain the inverse transform sampling for a slightly different distribution. With our algorithm, we were able to train NeRF end-to-end without the auxiliary task and improve the reconstruction quality. We achieve this by back-propagating the gradients through the sampler, and show that the original sampling algorithm fails to achieve similar quality in the same setup.

Below, Section 2 gives a recap of neural radiance fields. Then we proceed to the main contributions of our work in Section 3, namely the rendering algorithm fueled by Monte Carlo estimates and the novel sampling procedure. In Section 4 we discuss related work. In Subsection 5.1, we use our sampling algorithm to improve the hierarchical sampling scheme proposed for training NeRF. Finally, in Subsection 5.2 we apply the proposed Monte Carlo estimate to replace the standard rendering algorithm. With an efficient neural radiance field architecture, our algorithm decreases time per training iteration at the cost of reduced reconstruction quality. We also show that our Monte Carlo estimate can be used during inference of a pretrained model with no additional fine-tuning needed, and it can achieve better reconstruction quality at the same speed in comparison to the standard algorithm. Our source code is available at `https://github.com/GreatDrake/reparameterized-volume-sampling`.

2 NEURAL RADIANCE FIELDS

Neural radiance fields represent 3D scenes with a non-negative scalar density field $\sigma : \mathbb{R}^3 \to \mathbb{R}^+$ and a vector radiance field $c : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$. Scalar field σ represents volume density at each spatial location $\boldsymbol{x}$, and $c(\boldsymbol{x}, \boldsymbol{d})$ returns the light emitted from spatial location $\boldsymbol{x}$ in direction $\boldsymbol{d}$ represented as a normalized three dimensional vector.

For novel view synthesis, NeRF (Mildenhall et al., 2020) adapts a volume rendering algorithm that computes pixel color $C(\boldsymbol{r})$ as the expected radiance for a ray $\boldsymbol{r} = \boldsymbol{o} + t\boldsymbol{d}$ passing through a pixel from origin $\mathbf{o} \in \mathbb{R}^3$ in a direction $\mathbf{d} \in \mathbb{R}^3$. For ease of notation, we will denote density and radiance restricted to a ray $\boldsymbol{r}$ as

$$\sigma_{\boldsymbol{r}}(t) := \sigma(\boldsymbol{o} + t\boldsymbol{d}) \text{ and } c_{\boldsymbol{r}}(t) := c(\boldsymbol{o} + t\boldsymbol{d}, \boldsymbol{d}). \quad (1)$$

With that in mind, the expected radiance along ray $\boldsymbol{r}$ is given as

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} p_{\boldsymbol{r}}(t) c_{\boldsymbol{r}}(t) \mathrm{d}t, \quad (2)$$

where

$$p_{\boldsymbol{r}}(t) := \sigma_{\boldsymbol{r}}(t) \exp\left(- \int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s) \mathrm{d}s \right). \quad (3)$$

Here, t_n and t_f are *near* and *far* ray boundaries, and $p_{\boldsymbol{r}}(t)$ is an unnormalized probability density function of a random variable t on a ray $\boldsymbol{r}$. Intuitively, t is the location on a ray where the portion of light coming into the point $\boldsymbol{o}$ was emitted.

To approximate the nested integrals in Eq. 2, Max (1995) proposed to replace fields $\sigma_{\boldsymbol{r}}$ and $c_{\boldsymbol{r}}$ with a piecewise approximation on a grid $t_n = t_0 < t_1 < \cdots < t_m = t_f$ and compute the formula in Eq. 2 analytically for the approximation. In particular, a piecewise constant approximation with density σ_i and radiance c_i within i-th bin $[t_{i+1}, t_i]$ of width $\delta_i = t_{i+1} - t_i$ yields formula

$$\hat{C}(\boldsymbol{r}) = \sum_{i=1}^{m} w_i c_i, \quad (4)$$

where the weights are given by

$$w_i = (1 - \exp(-\sigma_i \delta_i)) \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right). \quad (5)$$

Importantly, Eq. 4 is fully differentiable and can be used as a part of a gradient-based learning pipeline. To reconstruct a scene NeRF runs a gradient based optimizer to minimize MSE between the predicted color and the ground truth color averaged across multiple rays and multiple viewpoints.

While the above approximation works in practice, it involves multiple evaluations of c and σ along a dense grid. Besides that, a ray typically intersects a solid surface at some point $t \in [t_n, t_f]$. In this case, probability density $p_{\boldsymbol{r}}(t)$ will concentrate its mass near t and, as a result, most of the terms in Eq. 4 will make a negligible contribution to the sum. To approach this problem, NeRF employs a hierarchical sampling scheme. Two networks are trained simultaneously: coarse (or proposal) and fine. Firstly, the coarse network is evaluated on a uniform grid of N_c points and a set of weights w_i is calculated as in Eq. 5. Normalizing these weights produces a piecewise constant PDF along the ray. Then N_f samples are drawn from this distribution and the union of the first and second sets of points is used to evaluate the fine network and compute the final color estimation. The coarse network is also trained to predict ground truth colors, but the color estimate for the coarse network is calculated only using the first set of N_c points.

3 REPARAMETERIZED VOLUME SAMPLING AND RADIANCE ESTIMATES

3.1 Reparameterized Expected Radiance Estimates

Monte Carlo methods give a natural way to approximate the expected color. For example, given k i.i.d. samples $t_1, \ldots, t_k \sim p_{\boldsymbol{r}}(t)$ and the normalizing constant $y_f := \int_{t_n}^{t_f} p_{\boldsymbol{r}}(t)\mathrm{d}t$, the sum

$$\hat{C}_{MC}(\boldsymbol{r}) = \frac{y_f}{k} \sum_{i=1}^{k} c_{\boldsymbol{r}}(t_i) \tag{6}$$

is an unbiased estimate of the expected radiance in Eq. 2. Moreover, samples $t_1, \ldots, t_k$ belong to high-density regions of $p_{\boldsymbol{r}}$ by design, thus for a degenerate density $p_{\boldsymbol{r}}$ even a few samples would provide an estimate with low variance. Importantly, unlike the approximation in Eq. 4, the Monte Carlo estimate depends on scene density σ implicitly through sampling algorithm and requires a custom gradient estimate for the parameters of σ. We propose a principled end-to-end differentiable algorithm to generate samples from $p_{\boldsymbol{r}}(t)$.

Our solution is primarily inspired by the reparameterization trick (Kingma and Ba, 2014; Rezende et al., 2014). We change the variable in Eq. 2. For $F_{\boldsymbol{r}}(t) := 1 - \exp\left(-\int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s\right)$ and $y := F_{\boldsymbol{r}}(t)$ we rewrite

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} c_{\boldsymbol{r}}(t) p_{\boldsymbol{r}}(t)\mathrm{d}t \tag{7}$$

$$= \int_{y_n}^{y_f} c_{\boldsymbol{r}}(F_{\boldsymbol{r}}^{-1}(y))\mathrm{d}y \tag{8}$$

$$= \int_{0}^{1} y_f c_{\boldsymbol{r}}(F_{\boldsymbol{r}}^{-1}(y_f u))\mathrm{d}u. \tag{9}$$

The integral boundaries are $y_n := F_{\boldsymbol{r}}(t_n) = 0$ and $y_f := F_{\boldsymbol{r}}(t_f)$. Function $F_{\boldsymbol{r}}(t)$ acts as the cumulative distribution function of the variable t with a single exception that, in general, $F_{\boldsymbol{r}}(t_f) \neq 1$. In volume rendering, $F_{\boldsymbol{r}}(t)$ is called opacity function with y_f being equal to overall pixel opaqueness. After the first change of variables in Eq. 8, the integral boundaries depend on opacity $F_{\boldsymbol{r}}$ and, as a consequence, on ray density $\sigma_{\boldsymbol{r}}$. We further simplify the integral by changing the integration boundaries to $[0, 1]$ and substituting $y_n = 0$.

Given the above derivation, we construct *the reparameterized Monte Carlo estimate* for the right-hand side integral in Eq. 9

$$\hat{C}^{R}_{MC}(\boldsymbol{r}) := \frac{y_f}{k} \sum_{i=1}^{k} c_{\boldsymbol{r}}(F_{\boldsymbol{r}}^{-1}(y_f u_i)), \tag{10}$$

with k i.i.d. $U[0, 1]$ samples $u_1, \ldots, u_k$. It is easy to show that the estimate in Eq. 10 is an unbiased estimate of expected color in Eq. 2 and its gradient is an unbiased estimate of the gradient of the expected color $C(\boldsymbol{r})$. Additionally, we propose to replace the uniform samples $u_1, \ldots, u_k$ with uniform independent samples within regular grid bins $v_i \sim U[\frac{i-1}{k+1}, \frac{i}{k+1}], i = 1, \ldots, k$. The latter samples yield a stratified variant of the estimate in Eq. 10 and, most of the time, lead to lower variance estimates (see Appendix B).

In the above estimate, random samples $u_1, \ldots, u_k$ do not depend on volume density $\sigma_{\boldsymbol{r}}$ or color $c_{\boldsymbol{r}}$. Essentially, for the reparameterized Monte Carlo estimate we generate samples from $p_{\boldsymbol{r}}(t)$ using inverse cumulative distribution function $F_{\boldsymbol{r}}^{-1}(y_f u)$. In what follows, we coin the term *reparameterized volume sampling (RVS)* for the sampling procedure. However, in practice, we cannot compute $F_{\boldsymbol{r}}$ analytically and can only query $\sigma_{\boldsymbol{r}}$ at certain ray points. Thus, in the following section, we introduce approximations of $F_{\boldsymbol{r}}$ and its inverse.

3.2 Opacity Approximations

The expected radiance estimate in Eq. 10 relies on opacity $F_{\boldsymbol{r}}(t) = 1 - \exp\left(-\int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s\right)$ and its inverse

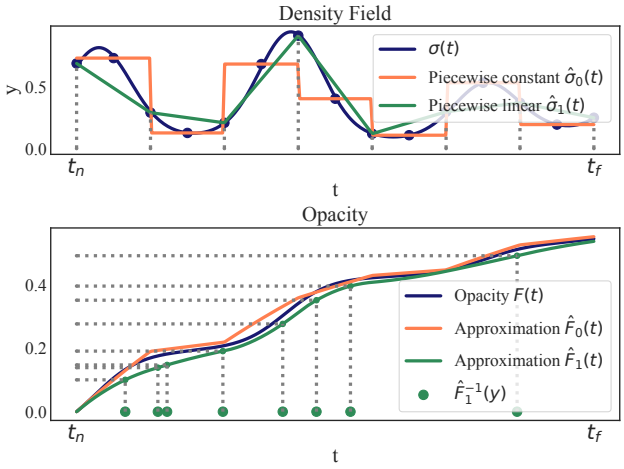

Figure 2: Illustration of opacity inversion. On the left, we approximate density field $\sigma_{\boldsymbol{r}}$ with a piecewise constant and a piecewise linear approximation. On the right, we approximate opacity $F_{\boldsymbol{r}}(t)$ and compute $F_{\boldsymbol{r}}^{-1}(y_f u)$ for $u \sim U[0,1]$.

$F_{\boldsymbol{r}}^{-1}(y)$. We propose to approximate the opacity using a piecewise density field approximation. Fig. 2 illustrates the approximations and ray samples obtained through opacity inversion. To construct the approximation, we take a grid $t_n = t_0 < t_1 < \cdots < t_m = t_f$ and construct either a piecewise constant or a piecewise linear approximation. In the former case, we pick a point within each bin $t_i \leq \hat{t}_i \leq t_{i+1}$ and approximate density with $\sigma_{\boldsymbol{r}}(\hat{t}_i)$ inside the corresponding bin. In the latter case, we compute $\sigma_{\boldsymbol{r}}$ in the grid points and interpolate the values between the grid points. Importantly, for a non-negative field these two approximations are also non-negative. Then we compute $\int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s$, which is as a sum of rectangular areas in the piecewise constant case

$$I_0(t) = \sum_{j=1}^{i} \sigma_{\boldsymbol{r}}(\hat{t}_j)(t_j - t_{j-1}) + \sigma_{\boldsymbol{r}}(\hat{t}_i)(t - t_i). \quad (11)$$

Analogously, the integral approximation $I_1(t)$ in the piecewise linear case is a sum of trapezoidal areas.

Given these approximations, we can approximate y_f and $F_{\boldsymbol{r}}$ in Eq. 10. We generate samples on a ray based on inverse opacity $F_{\boldsymbol{r}}^{-1}(y)$ by solving the equation

$$y_f u = F_{\boldsymbol{r}}(t) = 1 - \exp\left(-\int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s\right) \quad (12)$$

for t, where $u \in [0,1]$ is a random sample. We rewrite the equation as $-\log(1 - y_f u) = \int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s$ and note that integral approximations $I_0(t)$ and $I_1(t)$ are monotonic piecewise linear and piecewise quadratic functions. We obtain the inverse by first finding the bin that contains the solution and then solving a linear or a quadratic equation. Crucially, the solution t can be seen as a differentiable function of density field $\sigma_{\boldsymbol{r}}$ and we can back-propagate the gradients w.r.t. $\sigma_{\boldsymbol{r}}$ through t. We provide explicit formulae for t for both approximations in Appendix A.1 and discuss the solutions crucial for the numerical stability in Appendix A.2. In Appendix A.3, we provide the algorithm implementation and draw parallels with earlier work. Additionally, in Appendix A.4 we discuss an alternative approach to calculating inverse opacity and its gradients. We use piecewise linear approximations in Subsection 5.1 and piecewise constant in Subsection 5.2.

3.3 Application to Hierarchical Sampling

Finally, we propose to apply our RVS algorithm to the hierarchical sampling scheme originally proposed in NeRF. Here we do not change the final color approximation, utilizing the original one (Eq. 4), but modify the way the coarse density network is trained. The method we introduce consists of two changes to the original scheme. Firstly, we replace sampling from piecewise constant PDF along the ray defined by weights w_i (see Section 2) with our RVS sampling algorithm that uses piecewise linear approximation of $\sigma_{\boldsymbol{r}}$ and generates samples from $p_r(t)$ using inverse CDF. Secondly, we remove the auxiliary reconstruction loss imposed on the coarse network. Instead, we propagate gradients through sampling. This way, we eliminate the need for auxiliary coarse network losses and train the network to solve the actual task of our interest: picking the best points for evaluation of the fine network. All components of the model are trained together end-to-end from scratch. In Subsection 5.1, we refer to the coarse network as the proposal network, since such naming better captures its purpose.

4 RELATED WORK

Monte Carlo estimates for integral approximations. In this work, we revisit the algorithm introduced to approximate the expected color in Max (1995). Currently, it is the default solution in multiple works on neural radiance fields. Max (1995) approximate density and radiance fields with a piecewise constant functions along a ray and compute Eq. 2 as an approximation. Instead, we reparameterize Eq. 2 and construct Monte Carlo estimates for the integral. To compute the estimates in practice we use piecewise approximations only for the density field. The cumulative density function (CDF) used in our estimates involves integrating the density field along a ray. Lindell et al. (2021) construct field anti-derivatives to accelerate inference. While they use the anti-derivatives to compute 2 on a grid with

fewer knots, the anti-derivatives can be applied in our sampling method based on the inverse CDF without resorting to piecewise approximations.

In the past decade, integral reparameterizations have become a common practice in generative modeling (Kingma and Welling, 2013; Rezende et al., 2014) and approximate Bayesian inference (Blundell et al., 2015; Gal and Ghahramani, 2016; Molchanov et al., 2017). Similar to Equation 2, objectives in these areas require optimizing expected values with respect to distribution parameters. We refer readers to Mohamed et al. (2020) for a systematic overview. Notably, in computer graphics, Loubet et al. (2019) apply reparameterization to estimate gradients of path-traced images with respect to scene parameters.

NeRF acceleration through architecture and sparsity. Since the original NeRF work (Mildenhall et al., 2020), a number of approaches that aim to improve the efficiency of the model have been proposed. One family of methods tries to reduce the time required to evaluate the field. It includes a variety of architectures combining Fourier features (Tancik et al., 2020) and grid-based features (Garbin et al., 2021; Sun et al., 2022; Fridovich-Keil et al., 2022; Reiser et al., 2021). Besides grids, some works exploit space partitions based on Voronoi diagrams (Rebain et al., 2021), trees (Hu et al., 2022; Yu et al., 2021) and even hash tables (Müller et al., 2022). These architectures generally trade-off inference speed for parameter count. TensorRF (Chen et al., 2022) stores the grid tensors in a compressed format to achieve both high compression and fast performance. On top of that, skipping field queries for the empty parts of a scene additionally improves rendering time (Levoy, 1990). Recent works (such as Hedman et al. (2021); Fridovich-Keil et al. (2022); Liu et al. (2020); Li et al. (2022); Sun et al. (2022); Müller et al. (2022)) manually exclude low-weight components in Eq 4 to speed up rendering during training and inference. Below, we show that our Monte Carlo algorithm is compatible with fast architectures and sparse density fields, achieving comparable speedups by using a few radiance evaluations.

Anti-aliased scene representations. Mip-NeRF (Barron et al., 2021), Mip-NeRF 360 (Barron et al., 2022) and a more recent Zip-NeRF (Barron et al., 2023) represent a line of work that modifies scene representations. Relevant to our research is the fact that these models employ modifications of the original hierarchical sampling scheme, where the coarse network parameterizes some density field. Mip-NeRF parameterizes the coarse and the fine fields by the same neural network that represents the scene at a continuously-valued scale. Mip-NeRF 360 and Zip-NeRF use a separate model for proposal density, but train it to mimic the fine density rather than independently reconstructing the image. This means that our method for training the proposal density field can be potentially used to improve the performance of these models and simplify the training algorithm.

Algorithms for picking ray points. Mildenhall et al. (2020) employs a hierarchical scheme to generate ray points using an auxiliary density and color fields. Since then, a number of other methods for picking ray points, which focus on real-time rendering and aim to improve the efficiency of NeRF, have been proposed. DoNeRF (Neff et al., 2021) uses a designated depth oracle network supervised with ground truth depth maps. TermiNeRF (Piala and Clark, 2021) foregoes the depth supervision by distilling the sampling network from a pre-trained NeRF model. NeRF-ID (Arandjelović and Zisserman, 2021) adds a separate differentiable proposer neural network to the original NeRF model that maps outputs of the coarse network into a new set of samples. The model is trained in a two-stage procedure together with NeRF. The authors of NeuSample (Fang et al., 2021) use a sample field that directly transforms rays into point coordinates. The sample field can be further fine-tuned for rendering with a smaller number of samples. AdaNeRF (Kurz et al., 2022) proposes to use a sampling and a shading network. Samples from the sampling network are processed by the shading network that tries to predict the importance of samples and cull the insignificant ones. One of the key merits of our approach in comparison to these works is its simplicity. We simplify the original NeRF training procedure, while other works only build upon it, adding new components, training stages, constraints, or losses. Moreover, the absence of reliance on additional neural network components (not responsible for density or radiance) for sampling makes our approach better suited for fast NeRF architectures. Finally, our approach is suitable for end-to-end training of NeRF models from scratch, whereas the works mentioned above use pre-trained NeRF models or multiple training stages.

5 EXPERIMENTS

5.1 End-to-end Differentiable Hierarchical Sampling

In this section, we evaluate the proposed approach to hierarchical volume sampling (see Subsection 3.3).

Experimental setup. We do the comparison by fixing some training setup and training two models from scratch: one NeRF model is trained using the procedure proposed in Mildenhall et al. (2020) (further denoted as NeRF in the results), and the other one is trained using our modification described in Subsection 3.3 (further

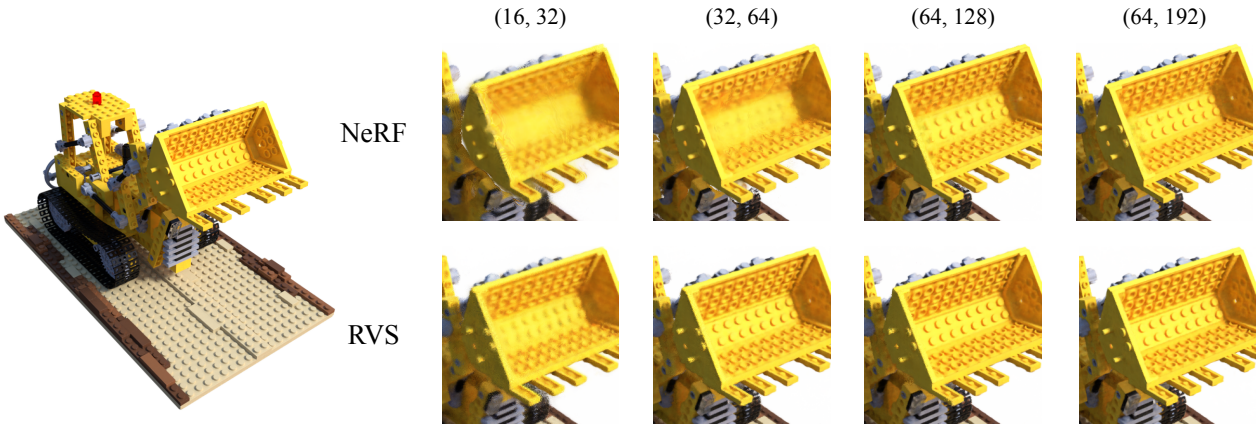

Figure 3: Comparison between renderings of test-set view on the Lego scene (Blender). Rows correspond to different (N_p, N_f) configurations. NeRF baseline has difficulties in reconstructing fine details for $(16, 32)$ and $(32, 64)$ configurations, and some parts remain blurry even in $(64, 128)$ and $(64, 192)$ configurations, while our method already produces realistic reconstruction for $(32, 64)$ configuration.

denoted as RVS). For both models, the final color approximation is computed as in Eq. 4, so the difference only appears in the proposal component.

We train all models for 500k iterations using the same hyperparameters as in the original paper with minor differences. We replace ReLU density output activation with Softplus. The other difference is that we use a smaller learning rate for the proposal density network in our method (start with 5×10^{-5} and decay to 5×10^{-6}) but the same for the fine network (start with 5×10^{-4} and decay to 5×10^{-5}). This is done since we observed that decreasing the proposal network learning rate improves stability of our method. We run the default NeRF training algorithm in our experiments with the same learning rates for proposal and fine networks. Further in the ablation study we show that decreasing the proposal network learning rate only degrades the performance of the base algorithm. We use PyTorch implementation of NeRF (Yen-Chen, 2020) in our experiments.

Comparative evaluation. We start the comparison on the Lego scene of the synthetic Blender dataset (Mildenhall et al., 2020) for different (N_p, N_f) configurations that correspond to the number of proposal and fine network evaluations. Note that this N_p, N_f notation does not directly correspond to the N_c, N_f notation used in Section 2 since the original NeRF model evaluates the fine network in $N_c + N_f$ points. For more details on training configurations and options for picking points for fine network evaluation, see Appendix D. The results are presented in Table 1. Our method outperforms the baseline across all configurations and all metrics, with the only exception of LPIPS in $(64, 192)$ configuration, where it showed similar performance. We observe that the improvement is more significant for smaller (N_p, N_f). Our method also has a minor speedup over the baseline due to the fact that the former does not use the radiance component of the proposal network. Fig. 3 in visualizes test-set view renderings of models trained by two methods.

Table 1: Comparison on the Lego scene of Blender dataset between NeRF training algorithm and our modification depending on the number of proposal and fine network evaluations per ray (N_p, N_f). The $(64, 192)$ configuration is the one originally used in NeRF. The training time column depicts relative training time on a single NVIDIA A100 GPU (1.0 being 12 hours).

	Evals.		Train	PSNR ↑	SSIM ↑	LPIPS ↓
	N_p	N_f	time			
NeRF	16	32	0.39	27.09	0.913	0.121
RVS	16	32	0.37	**29.18**	**0.928**	**0.112**
NeRF	32	64	0.52	30.11	0.947	0.070
RVS	32	64	0.48	**31.89**	**0.955**	**0.066**
NeRF	64	128	0.79	32.14	0.958	0.053
RVS	64	128	0.76	**32.80**	**0.963**	**0.051**
NeRF	64	192	1.0	32.69	0.962	**0.048**
RVS	64	192	0.98	**33.03**	**0.964**	**0.047**

We also visualize proposal and fine densities learned by two algorithms in Fig. 4. Figures are constructed by fixing some value of z coordinate and calculating the density on (x, y)-plane. While fine density visualizations look similar, proposal densities turn out very different. This happens due to the fact that the original algorithm trains the proposal network to reconstruct the scene (but using a smaller number of points for color estimation), while our algorithm trains this network to sample points for fine network evaluation that

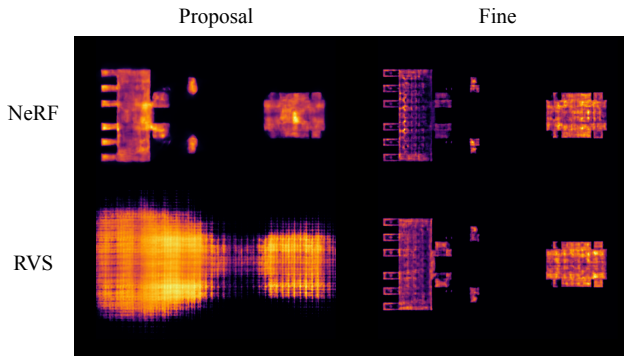

Figure 4: Visualizations across 2D slice of proposal and fine densities learnt on the Lego scene in $(32, 64)$ configuration. Brighter pixels correspond to larger density values.

Table 2: Comparison with NeRF on Blender and LLFF datasets in $(N_p = 32, N_f = 64)$ configuration.

	Blender Dataset		
	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	29.49	0.934	0.085
RVS	**30.26**	**0.939**	**0.082**
	LLFF Dataset		
	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	26.01	0.796	0.273
RVS	**26.24**	**0.799**	**0.270**

would lead to a better overall reconstruction. Since RVS shows better reconstruction quality, and its proposal densities are significantly different from the NeRF ones, one can argue that a proposal density that tries to mimic the fine density is not the optimal choice.

Next, we evaluate our method on all scenes from Blender, as well as the LLFF (Mildenhall et al., 2019) dataset containing real scenes. Table 2 depicts the results averaged across all scenes. The results for individual scenes can be found in Appendix F. Our approach shows improvement over the baseline on all scenes of Blender dataset. While the improvement is less pronounced on LLFF dataset, it is still present across all metrics on average. Fig. 7 in Appendix E visually depicts the quality of reconstruction on T-Rex scene.

Unbounded scenes. In addition, we evaluate our approach with NeRF++ (Zhang et al., 2020) modification designed for unbounded scenes. NeRF++ utilizes the same hierarchical scheme as the original NeRF, so we ran the same setup as previously: one model is trained using the original procedure, and for the other one we replace the sampling algorithm with RVS and propagate gradients through sampling instead of using a separate reconstruction loss for the proposal network. We did not modify any hyperparameters in comparison to Zhang et al. (2020) apart from using a smaller (N_p, N_f) configuration and a smaller proposal learning rate in our approach, the same as in the previous experiments. We run the comparison on LF (Yücer et al., 2016) and T&T (Knapitsch et al., 2017) datasets containing unbounded real scenes. The results are presented in Table 3. Our approach also shows improvement over NeRF++ across all metrics on both datasets. Table 12 and Table 13 in Appendix F present the results for individual scenes.

Table 3: Comparison with NeRF++ on LF and T&T datasets in $(N_p = 32, N_f = 64)$ configuration.

	LF Dataset		
	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF++	23.99	0.784	0.287
RVS	**24.63**	**0.812**	**0.253**
	T&T Dataset		
	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF++	19.21	0.612	0.493
RVS	**19.62**	**0.622**	**0.472**

Table 4: Ablation study with NeRF on Blender dataset in $(N_p = 32, N_f = 64)$ configuration.

	Blender Dataset		
	PSNR ↑	SSIM ↑	LPIPS ↓
aux. loss (base NeRF)	29.49	0.934	0.085
aux. loss (smaller prop. lr)	29.23	0.932	0.090
aux. loss (union of points)	29.06	0.929	0.096
aux. loss (our sampling)	29.70	0.936	**0.082**
end-to-end (NeRF sampling)	29.07	0.929	0.095
end-to-end (our sampling)	**30.26**	**0.939**	**0.082**

Ablation study. Finally, we ablate the influence of some components of our approach on the results. Table 4 presents the ablation study. Firstly, we run NeRF baseline with a decreased proposal network learning rate, thus fully matching all training hyperparameters with our method. This only reduces all metrics. Next, we run our approach that trains the proposal network end-to-end, but replace our algorithm that draws samples from the proposal distribution with the sampling algorithm originally proposed in NeRF. Even though the original work does not propagate gradients through sampling, the algorithm is still end-to-end differentiable, thus the setup is plausible. It also produces results that fall behind the baseline NeRF. This shows that while both algorithms are differentiable, ours is better suited for end-to-end optimization. We discuss some possible reasons behind these results and the differences between the two algorithms in Appendix A.3. After that, we run the original NeRF approach that uses an auxiliary loss for proposal network training, but replace the sampling algorithm with ours (without propagating gradients through sampling). This variant performs better than the baseline, but still falls behind

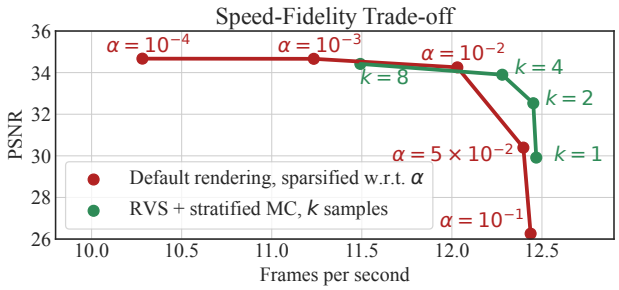

Figure 5: Test-time rendering quality as a function of rendering speed. Given a scene representation pre-trained with the standard rendering algorithm, we evaluate rendering algorithms at various configurations.

end-to-end training with RVS. Finally, we run the baseline with a different strategy for picking fine points (see Appendix D for a detailed discussion), which also leads to degradation of the baseline performance.

5.2 Scene Reconstruction with Monte Carlo Estimates

In this section, we evaluate the proposed Monte Carlo radiance estimate (see Subsection 3.1 and Eq. 10) as a part of the rendering algorithm for scene reconstruction. Given an accurate density field approximation, our color estimate is unbiased for any given number of samples k. Therefore, our estimate is especially suitable for architectures that can evaluate the density field faster than the radiance field. As an example, we pick a voxel-based radiance field model DVGO (Sun et al., 2022) that parametrizes density field as a voxel grid and relies on a combination of voxel grid and a view-dependent neural network to parameterize radiance. In Appendix C, we evaluate the inference time of the model to illustrate the benefits of our approach. In our experiments, we take the default model parameters and only replace the rendering algorithm.

Experimental setup & comparative evaluation. In Table 5, we report iterations per second and peak memory consumption on the Lego scene of Blender dataset during training. One of the primary goals of DVGO is to reduce the training time of a scene model. To achieve that goal, the authors mask components in the sum in Eq. 4 that have weights below a certain threshold α. We achieve a similar effect without thresholding: $k = 8$ samples yield a comparable $\times 6$ speedup and with fewer samples, we reduce iteration time even further. However, the speed-up in our estimate comes at the cost of additional estimate variance.

Next, we evaluate the effect of additional variance on the rendering fidelity. In Fig. 5, we report test PSNR on a fixed pre-trained Lego scene representation for both rendering algorithms, comparing them at the *inference stage*. Our algorithm achieves the same average PSNR with $k = 8$ samples and outperforms the default rendering algorithm in case of a higher sparsity threshold α.

Table 5: Training iteration times and peak memory consumption for different color approximations.

DVGO Renderer	Speed	Memory
Default w/o sparisty	24 it/s	9 GB
Default w/ sparsity	160 it/s	5 GB
MC + RVS, $k = 128$	20 it/s	8 GB
MC + RVS, $k = 8$	170 it/s	5 GB
MC + RVS, $k = 1$	260 it/s	5 GB

Table 6: Scene reconstruction results on Blender for training with Monte Carlo estimates.

DVGO Renderer	Train time (min)	PSNR↑	SSIM↑	LPIPS↓
Default w/ sparsity	2:48	**31.90**	**0.956**	**0.054**
MC + RVS, $k = 4$	2:24	31.19	0.951	0.059
MC + RVS, $k = 8$	2:00	31.13	0.951	0.059
MC + RVS, adaptive k	2:54	31.44	0.953	0.056

In Table 6, we report the performance of models trained with various color estimates (see Appendix F for per-scene results), comparing them at the *training stage*. We train a model with $k = 4$ samples using $\times\frac{3}{2}$ more training steps than with $k = 8$ samples aiming to achieve similar training times. Additionally, we consider a model that chooses the number of samples k on each ray adaptively between 4 and 48 based on the number of grid points with high density. The adaptive number of samples allows to reduce estimate variance without a drastic increase in training time. We evaluate our models with $k = 64$ samples to mitigate the effect of variance on evaluation.

As Table 6 indicates, the training algorithm with our color approximation fails to outperform the base algorithm in terms of reconstruction quality; however, it allows for the faster training of the model.

Ablation study. We ablate the proposed algorithm on the Lego scene to gain further insights into the difference in reconstruction fidelity when it is used for training. Specifically, we ablate parameters affecting optimization aiming to match the reconstruction quality of the default algorithm. The results are given in Table 7. The increase in training steps or the number of samples improves the performance but does not lead to matching results. Increased spline density improves both our model and the baseline to the same extent. We also noticed that the standard objective estimates the expected loss $\mathbb{E}L_2(\hat{C}, C_{gt})$ rather than the loss at expected radiance $L_2(\mathbb{E}\hat{C}, C_{gt})$, which leads to

Table 7: Ablation study for training with Monte Carlo estimates on the Lego scene.

Ablated Feature	PSNR
MC + RVS with adaptive k	33.85
Training steps ($\times 7$)	+0.28
Monte Carlo samples ($\times 2$ on avg.)	+0.15
Dense grid on rays ($\times 2$)	+0.29
Unbiased loss $L_2(\mathbb{E}\hat{C}, C_{gt})$	+0.25
Density grid resolution ($160^3 \to 256^3$)	+0.63
Radiance grid resolution ($160^3 \to 256^3$)	+0.08
σ + c resolution ($160^3 \to 256^3$)	**+0.86**
DVGO	34.64

an additional bias towards low-variance densities. The estimate $(\hat{C}_1 - C_{gt})(\hat{C}_2 - C_{gt})$ with two i.i.d. color estimates is an unbiased estimate of the latter and allows reconstructing non-degenerate density fields, but the estimate has little effect in case of degenerate densities omnipresent in 3D scenes. Finally, with a denser ray grid our algorithm surpasses the baseline PSNR at the cost of increased training time.

To summarize, training with the reparameterized Monte Carlo estimate currently does not fully match the fidelity of the standard approach. At the same time, Monte Carlo radiance estimates provide a straightforward mechanism to control both training and inference speed.

6 CONCLUSION

The core of our contribution is an end-to-end differentiable ray point sampling algorithm. We utilize it to construct an alternative rendering algorithm based on Monte Carlo, which provides an explicit mechanism to control rendering time during the inference and training stages. While it is able to outperform the standard rendering algorithm at the inference stage given a pre-trained model, it achieves lower reconstruction quality when used during training, which suggests areas for future research. At the same time, we show that the proposed sampling algorithm improves scene reconstruction in hierarchical models and simplifies the training approach by disposing of auxiliary losses.

Acknowledgements

The study was carried out within the strategic project "Digital Transformation: Technologies, Effects and Performance", part of the HSE University "Priority 2030" Development Programme. The results on hierarchical sampling from Section 5.1 were obtained by Kirill Struminsky with the support of the grant for research centers in the field of AI provided by the Analytical Center for the Government of the Russian Federation (ACRF) in accordance with the agreement on the provision of subsidies (identifier of the agreement 000000D730321P5Q0002) and the agreement with HSE University No. 70-2021-00139. This research was supported in part through computational resources of HPC facilities at HSE University (Kostenetskiy et al., 2021).

References

Arandjelović, R. and Zisserman, A. (2021). Nerf in detail: Learning to sample for view synthesis. *arXiv preprint arXiv:2106.05264*.

Barron, J. T., Mildenhall, B., Tancik, M., Hedman, P., Martin-Brualla, R., and Srinivasan, P. P. (2021). Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864.

Barron, J. T., Mildenhall, B., Verbin, D., Srinivasan, P. P., and Hedman, P. (2022). Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479.

Barron, J. T., Mildenhall, B., Verbin, D., Srinivasan, P. P., and Hedman, P. (2023). Zip-nerf: Anti-aliased grid-based neural radiance fields. *ICCV*.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR.

Chen, A., Xu, Z., Geiger, A., Yu, J., and Su, H. (2022). Tensorf: Tensorial radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*, pages 333–350. Springer.

Fang, J., Xie, L., Wang, X., Zhang, X., Liu, W., and Tian, Q. (2021). Neusample: Neural sample field for efficient view synthesis. *arXiv preprint arXiv:2111.15552*.

Figurnov, M., Mohamed, S., and Mnih, A. (2018). Implicit reparameterization gradients. *Advances in Neural Information Processing Systems*, 31.

Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., and Kanazawa, A. (2022). Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR.

Garbin, S. J., Kowalski, M., Johnson, M., Shotton, J., and Valentin, J. (2021). Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355.

Hedman, P., Srinivasan, P. P., Mildenhall, B., Barron, J. T., and Debevec, P. (2021). Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884.

Hu, T., Liu, S., Chen, Y., Shen, T., and Jia, J. (2022). Efficientnerf efficient neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12902–12911.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Knapitsch, A., Park, J., Zhou, Q.-Y., and Koltun, V. (2017). Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13.

Kostenetskiy, P., Chulkevich, R., and Kozyrev, V. (2021). Hpc resources of the higher school of economics. In *Journal of Physics: Conference Series*, volume 1740, page 012050. IOP Publishing.

Kurz, A., Neff, T., Lv, Z., Zollhöfer, M., and Steinberger, M. (2022). Adanerf: Adaptive sampling for real-time rendering of neural radiance fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII*, pages 254–270. Springer.

Levoy, M. (1990). Efficient ray tracing of volume data. *ACM Transactions on Graphics (TOG)*, 9(3):245–261.

Li, R., Tancik, M., and Kanazawa, A. (2022). Nerfacc: A general nerf acceleration toolbox. *arXiv preprint arXiv:2210.04847*.

Lindell, D. B., Martel, J. N., and Wetzstein, G. (2021). Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14556–14565.

Liu, L., Gu, J., Zaw Lin, K., Chua, T.-S., and Theobalt, C. (2020). Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663.

Loubet, G., Holzschuch, N., and Jakob, W. (2019). Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14.

Max, N. (1995). Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108.

Mildenhall, B., Srinivasan, P. P., Ortiz-Cayon, R., Kalantari, N. K., Ramamoorthi, R., Ng, R., and Kar, A. (2019). Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. (2020). Monte carlo gradient estimation in machine learning. *J. Mach. Learn. Res.*, 21(132):1–62.

Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR.

Müller, T., Evans, A., Schied, C., and Keller, A. (2022). Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15.

Neff, T., Stadlbauer, P., Parger, M., Kurz, A., Mueller, J. H., Chaitanya, C. R. A., Kaplanyan, A., and Steinberger, M. (2021). Donerf: Towards real-time rendering of compact neural radiance fields using depth oracle networks. In *Computer Graphics Forum*, volume 40, pages 45–59. Wiley Online Library.

Piala, M. and Clark, R. (2021). Terminerf: Ray termination prediction for efficient neural rendering. In *2021 International Conference on 3D Vision (3DV)*, pages 1106–1114. IEEE.

Rebain, D., Jiang, W., Yazdani, S., Li, K., Yi, K. M., and Tagliasacchi, A. (2021). Derf: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14161.

Reiser, C., Peng, S., Liao, Y., and Geiger, A. (2021). Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR.

Sun, C., Sun, M., and Chen, H.-T. (2022). Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5459–5469.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547.

Yen-Chen, L. (2020). Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/.

Yu, A., Li, R., Tancik, M., Li, H., Ng, R., and Kanazawa, A. (2021). Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761.

Yücer, K., Sorkine-Hornung, A., Wang, O., and Sorkine-Hornung, O. (2016). Efficient 3d object segmentation from densely sampled light fields with applications to 3d reconstruction. *ACM Transactions on Graphics (TOG)*, 35(3):1–15.

Zhang, K., Riegler, G., Snavely, N., and Koltun, V. (2020). Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes, see Section 2 and Section 3]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes, see Appendix B and Appendix C for numerical analysis]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes, see Section 1]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Yes, see Section 3 and Appendix A.1]
   (c) Clear explanations of any assumptions. [Yes, see Section 3]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes, see Section 1 and Section 5]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes, see Section 5]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [No. We use well-established measures to report the performance of the proposed algorithm. We report the measures across a variety of datasets and a range of hyperparameters. We do not include results averaged across multiple runs as randon reinitialization has almost no effect on the resulting performance.]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes, see Section 5]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A INVERSE OPACITY CALCULATION

A.1 Inverse Functions for Density Integrals

In this section, we derive explicit formulae for the density integral inverse used in inverse opacity.

A.1.1 Piecewise Constant Approximation Inverse

We start with a formula for the integral

$$I_0(t) = \sum_{j=1}^{i} \sigma_{\boldsymbol{r}}(\hat{t}_j)(t_j - t_{j-1}) + \sigma_{\boldsymbol{r}}(\hat{t}_i)(t - t_i) \tag{13}$$

and solve for t equation

$$y = I_0(t). \tag{14}$$

The equation above is a linear equation with solution

$$t = t_i + \frac{y - \sum_{j=1}^{i} \sigma_{\boldsymbol{r}}(\hat{t}_j)(t_j - t_{j-1})}{\sigma_{\boldsymbol{r}}(\hat{t}_i)}. \tag{15}$$

In our implementation we add small ϵ to the denominator to improve stability when $\sigma_{\boldsymbol{r}}(\hat{t}_i) \approx 0$.

A.1.2 Piecewise Linear Approximation Inverse

The piecewise linear density approximation yield a piecewise quadratic function

$$I_1(t) = \sum_{j=1}^{i} \frac{\sigma_{\boldsymbol{r}}(t_j) + \sigma_{\boldsymbol{r}}(t_{j-1})}{2}(t_j - t_{j-1}) + \frac{(\sigma_{\boldsymbol{r}}(t_i) + \bar{\sigma}_{\boldsymbol{r}}(t))}{2}(t - t_i), \tag{16}$$

where $\bar{\sigma}_{\boldsymbol{r}}(t) = \sigma_{\boldsymbol{r}}(t_i)\frac{t_{i+1}-t}{t_{i+1}-t_i} + \sigma_{\boldsymbol{r}}(t_{i+1})\frac{t-t_i}{t_{i+1}-t_i}$ is the interpolated density at t. Again, we solve

$$y = I_1(t) \tag{17}$$

for t. We change the variable to $\Delta t := t - t_i$ and note that terms a and c in quadratic equation

$$0 = a\Delta t^2 + b\Delta t + c \tag{18}$$

will be

$$a = \frac{\sigma_{\boldsymbol{r}}(t_{i+1}) - \sigma_{\boldsymbol{r}}(t_i)}{2} \tag{19}$$

$$c = \left(\sum_{j=1}^{i} \frac{\sigma_{\boldsymbol{r}}(t_j) + \sigma_{\boldsymbol{r}}(t_{j-1})}{2}(t_j - t_{j-1}) - y \right) \times (t_{i+1} - t_i) \tag{20}$$

and with a few algebraic manipulations we find the linear term

$$b = \sigma_{\boldsymbol{r}}(t_i) \times (t_{i+1} - t_i). \tag{21}$$

Since our integral monotonically increases, we can deduce that the root Δt must be

$$\Delta t = \frac{-b + \sqrt{b^2 - 4ac}}{2a}. \tag{22}$$

However, this root is computationally unstable when $a \approx 0$. The standard trick is to rewrite the root as

$$\Delta t = \frac{2c}{b + \sqrt{b^2 - 4ac}}. \tag{23}$$

For computational stability, we add small ϵ to the square root argument. See the supplementary notebook for details.

A.2 Numerical Stability in Inverse Opacity

Inverse opacity input y is a combination of a uniform sample u and ray opacity $y_f = 1 - \exp\left(-\int_{t_n}^{t_f} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s\right)$:

$$y = -\log(1 - y_f u). \tag{24}$$

The expression above is a combination of a logarithm and exponent. We rewrite it to replace with more reliable logsumexp operator:

$$y = -\log\left(\exp(\log(1-u)) + \exp(\log u - \int_{t_n}^{t_f} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s)\right). \tag{25}$$

In practice, for opaque rays $\int_{t_n}^{t_f} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s \approx 0$ implementation of logsumexp becomes computationally unstable. In this case, we replace y with a first order approximation $u \cdot \int_{t_n}^{t_f} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s$.

A.3 Parallels with Prior Work and Algorithm Implementation

Original NeRF architecture uses inverse transform sampling to generate a grid for a fine network. They define a distribution based on Eq. 5 with piecewise constant density. In turn, we do inverse transform sampling from a distribution induced by a piecewise interpolation of $\sigma_{\boldsymbol{r}}$ in Eq. 3. The two approximation approaches yield distinct sampling algorithms. In Listing 1, we provide a numpy implementation of the two algorithms to highlight the difference. We rewrite NeRF sampling algorithm in an equivalent simplified form to facilitate the comparison. For simplicity, we provide implementation of RVS only for a piecewise constant approximation of σ. The inversion algorithm described in Appendix A.1 is hidden under the hood of *np.interp()*.

The interpolation scheme in our algorithm can be seen as linear interpolation of density field rather than probabilities. As a result, samples follow the Beer-Lambert law within each bin. The law describes light absorption in a homogeneous medium. In contrast, the hierarchical sampling scheme in NeRF interpolates exponentiated densities. We speculate that the numerical instability of the exponential function makes the latter algorithm less suitable for end-to-end optimization (as observed in Table 4).

```python
import numpy as np

def inverse_cdf(u, sigmas, ts, sampling_mode):
    """ Get inverse CDF sample

    Arguments:
    u - array of uniform random variables
    sigmas - array of density values on a grid
    ts - array of grid knots
    """
    # Compute $\int_{t_0}^{t_i} \sigma(s) ds$
    bin_integrals = sigmas * np.diff(ts)
    prefix_integrals = np.cumsum(bin_integrals)
    prefix_integrals = np.concatenate([np.zeros(1), prefix_integrals])
    # Get inverse CDF argument
    rhs = -np.expm1(-bin_integrals.sum()) * u
    if sampling_mode == 'rvs':  # interpolate $\int \sigma(s) ds$
        return np.interp(-np.log1p(-rhs), prefix_integrals, ts)
    elif sampling_mode == 'nerf':  # interpolate CDF
        return np.interp(rhs, -np.expm1(-prefix_integrals), ts)
```

Listing 1: Numpy implementation of the inverse transform sampling procedure proposed in this work and the inverse transform sampling proposed in NeRF. The implementation assumes a single ray for brevity.

A.4 Implicit Inverse Opacity Gradients

To compute the estimates in Eq. 10, we need to compute the inverse opacity $F_{\boldsymbol{r}}^{-1}(y)$ along with its gradient. In the main paper, we invert opacity explicitly with a differentiable algorithm. Alternatively, we could invert $F_{\boldsymbol{r}}(t) = 1 - \exp\left(-\int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s)\mathrm{d}s\right)$ with binary search. This approach can be used in situations when the formula for inverse opacity cannot be explicitly derived.

Opacity $F_{\boldsymbol{r}}(t)$ is a monotonic function and for $y \in (y_n, y_f) = (F_{\boldsymbol{r}}(t_n), F_{\boldsymbol{r}}(t_f))$ the inverse lies in (t_n, t_f). To compute $F_{\boldsymbol{r}}^{-1}(y)$, we start with boundaries $t_l = t_n$ and $t_r = t_f$ and gradually decrease the gap between the boundaries based on the comparison of $F_{\boldsymbol{r}}(\frac{t_l+t_r}{2})$ with y. Importantly, such procedure is easy to parallelize across multiple inputs and multiple rays.

However, we cannot back-propagate through the binary search iterations and need a workaround to compute the gradient $\frac{\partial t}{\partial \theta}$ of $t(\theta) = F_{\boldsymbol{r}}^{-1}(y, \theta)$. To do this, we follow Figurnov et al. (2018) and compute differentials of the right and the left hand side of equation $y(\theta) = F_{\boldsymbol{r}}(t, \theta)$

$$\frac{\partial y}{\partial \theta} \mathrm{d}\theta = \frac{\partial F_{\boldsymbol{r}}}{\partial t} \frac{\partial t}{\partial \theta} \mathrm{d}\theta + \frac{\partial F_{\boldsymbol{r}}}{\partial \theta} \mathrm{d}\theta. \tag{26}$$

By the definition of $F_{\boldsymbol{r}}(t, \theta)$ we have

$$\frac{\partial F_{\boldsymbol{r}}}{\partial t} = (1 - F_{\boldsymbol{r}}(t, \theta)) \sigma_{\boldsymbol{r}}(t, \theta), \tag{27}$$

$$\frac{\partial F_{\boldsymbol{r}}}{\partial \theta} = (1 - F_{\boldsymbol{r}}(t, \theta)) \frac{\partial}{\partial \theta} \left(\int_{t_n}^{t} \sigma_r(s, \theta) \mathrm{d}s \right). \tag{28}$$

We solve Eq. 26 for $\frac{\partial t}{\partial \theta}$ and substitute the partial derivatives using Eqs. 27 and 28 to obtain the final expression for the gradient

$$\frac{\partial t}{\partial \theta} = \frac{\frac{\partial y}{\partial \theta} - (1 - F_{\boldsymbol{r}}(t, \theta)) \frac{\partial}{\partial \theta} \int_{t_n}^{t} \sigma_{\boldsymbol{r}}(s, \theta) \mathrm{d}s}{(1 - F_{\boldsymbol{r}}(t, \theta)) \sigma_{\boldsymbol{r}}(t, \theta)}. \tag{29}$$

Automatic differentiation can be used to compute $\partial y / \partial \theta$ and $\frac{\partial}{\partial \theta} \int_{t_n}^{t} \sigma(s) \mathrm{d}s$ to combine the results as in Eq. 29.

B RADIANCE ESTIMATES FOR A SINGLE RAY

In this section, we evaluate the proposed Monte Carlo radiance estimate (see Eq. 10) in a one-dimensional setting. In this experiment, we assume that we know density in advance and show how the estimate variance depends on the number of radiance calls. Compared to sampling approaches, the standard approximation from Eq. 4 has zero variance but does not allow controlling the number of radiance calls.

Our experiment models light propagation on a single ray in two typical situations. The upper row of Fig. 6 defines a scalar radiance field (orange) $c_{\boldsymbol{r}}(t)$ and opacity functions (blue) $F_{\boldsymbol{r}}(t)$ for *"Foggy"* and *"Wall"* density fields. The first models a semi-transparent volume, which often occurs after model initialization during training. In the second, light is emitted from a single point on a ray, which is common in applications.

For the two fields we estimated the expected radiance $C(\boldsymbol{r}) = \int_{t_n}^{t_f} c_{\boldsymbol{r}}(t) \mathrm{d}F_{\boldsymbol{r}}(t)$. We consider two baseline methods (both in red in Fig. 6): the first is a Monte Carlo estimate of C obtained with uniform distribution on a ray $U[t_n, t_f]$, and its stratified modification with a uniform grid $t_n = t_0 < \cdots < t_k = t_f$ (note that here we use k to denote the number of samples, not the number of grid points m in piecewise density approximation):

$$\hat{C}_{\mathrm{IW}}(\boldsymbol{r}) = \sum_{i=1}^{k} (t_i - t_{i-1}) c_{\boldsymbol{r}}(\tau_i) \left. \frac{\mathrm{d}F_{\boldsymbol{r}}}{\mathrm{d}t} \right|_{t=\tau_i}, \tag{30}$$

where $\tau_i \sim U[t_{i-1}, t_i]$ are independent uniform bin samples. We compare the baseline against the estimate from Eq. 10 and its stratified modification. All estimates are unbiased. Therefore, we only compare the estimates' variances for a varying number of samples m.

In all setups, our stratified estimate uniformly outperforms the baselines. For the more challenging "foggy" field, approximately $k = 32$ samples are required to match the baseline performance for $k = 256$. We match the baseline with only a $k = 4$ samples for the "wall" field. Inverse transform sampling requires only a few points for degenerate distributions.

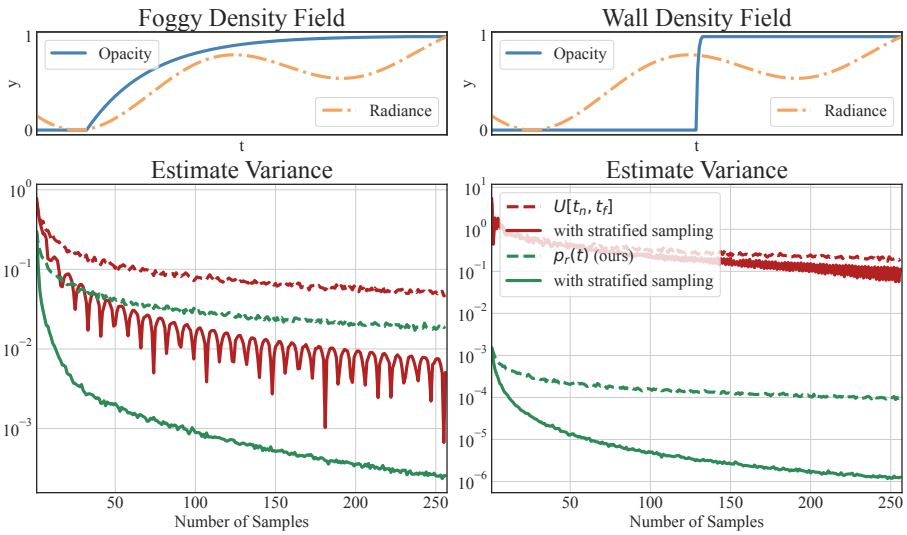

Figure 6: Color estimate variance compared for a varying number of samples. The upper plot illustrates underlying opacity function on a ray; the lower graph depicts variance in logarithmic scale. Compared to a naive estimate of the integral with uniform samples (dashed red), inverse transform sampling exhibits lower variance (dashed green). Stratified sampling improves variance in both setups (solid lines).

C COMPUTATIONAL EFFICIENCY OF APPROXIMATIONS

In this section, we analyze our approximation method's computational efficiency and compare it with the numerical approximation from Eq. 4. To illustrate it in a more practical setting, we determine the time performance of the estimates relative to the batch processing time of a radiance field model. As an example, we pick a recent voxel-based radiance field model DVGO Sun et al. (2022) that parameterizes density field as a voxel grid and combines voxel grid with a view-dependent neural network to obtain a hybrid parameterization of radiance.

Table 8: Computational complexity of three stages of color estimation.

	Voxel Grid (σ)	Hybrid (c)	$\mathbb{E}\hat{C}$
Baseline	0.00025s	0.02887s	0.00187s
Ours, 32	0.00025s	0.00419s	0.00369s
Ours, 64	0.00025s	0.00665s	0.00372s
Ours, 128	0.00025s	0.01334s	0.00381s
Ours, 256	0.00025s	0.02887s	0.00383s

Despite the toy nature of the experiment, we focus on a setting and hyperparameters commonly used in the rendering experiments. We take a batch of size 2048, draw 256 points along each of the corresponding rays and use them to calculate density σ. In the baseline setting, we then use the same points to calculate radiance c and estimate pixel colors with Eq. 4. In contrast to this pipeline, we propose to use calculated values of σ to make a piecewise constant approximation of density, generate a varying number (32, 64, 128, 256) of Monte Carlo samples and use them to calculate the stratified version of the color estimate (Eq 10). We parameterize σ with a voxel grid and c with a hybrid architecture used in DVGO with the default parameters.

In Table 8 we report time measurements for each of the stages of color estimation: calculating density field in 256 points along each ray, calculating radiance field in the Monte Carlo samples (or in the same 256 points in case of baseline) and sampling combined with calculating the approximation given σ and c (just calculating the approximation in case of baseline). All calculations were made on NVIDIA GeForce RTX 3090 Ti GPU and include both forward and backward passes.

First of all, σ computation time is equal for all of the cases and has order of 10^{-4} seconds, negligible in comparison with other stages. In terms of calculating the approximation, both baseline and our method work proportionally

to 10^{-3} seconds, but Monte Carlo estimate take 2.3 to 2.4 times more. Nevertheless, in the mentioned practical scenario this difference is not crucial, since computation of c, the heaviest part, takes up to 3×10^{-2} seconds. Even in the case of 256 radiance evaluations the difference in total computation time is less than 10%. This makes our method at least comparable with the baseline for architectures that can evaluate the density field faster than the radiance field. At the same time, our approach allows to explicitly control the number of radiance evaluations k, improving the computational efficiency even further given suitable architectures.

D PICKING FINE POINTS IN DIFFERENTIABLE HIERARCHICAL SAMPLING

Table 9: Comparison of various hierarchical sampling configurations on the Lego scene of Blender dataset.

	Evaluations		NeRF			RVS		
	N_p	N_f	PSNR ↑	SSIM ↑	LPIPS	PSNR ↑	SSIM ↑	LPIPS
Union	16	32	25.81	0.890	0.150	27.18	0.903	0.145
No union	16	32	**27.09**	**0.913**	**0.121**	**29.18**	**0.928**	**0.112**
Union	32	64	29.61	0.939	0.084	30.34	0.942	0.088
No union	32	64	**30.11**	**0.947**	**0.070**	**31.89**	**0.955**	**0.066**
Union	64	128	**32.14**	**0.958**	0.053	31.63	0.952	0.068
No union	64	128	31.87	0.958	0.054	**32.80**	**0.963**	**0.051**
Union	64	192	**32.69**	**0.962**	**0.048**	32.46	0.960	0.056
No union	64	192	32.14	0.960	0.051	**33.03**	**0.964**	**0.047**

We consider two options for picking N_f points for fine network evaluation: either take N_f samples from the proposal distribution (first option), or take the union of N_p grid points that were used to construct the distribution and $N_f - N_p$ new samples from the proposal distribution (second option, originally used in NeRF). In Table 1, we report the best result out of the two options, and Table 9 presents the results for both options. Our method works best with the first option. The baseline performs better with the first option for $(16, 32)$, $(32, 64)$ configurations and with the second option for other configurations. Comparisons in Table 2 are done in $(N_p = 32, N_f = 64)$ configuration with the first option. We use this configuration for comparison as both methods perform best with the same option and produce better results than in $(N_p = 16, N_f = 32)$ configuration. In the ablation study in Table 4, we also show that the base model works worse with the second option in $(N_p = 32, N_f = 64)$ configuration on average across other scenes from Blender dataset.

E ADDITIONAL VISUALIZATIONS

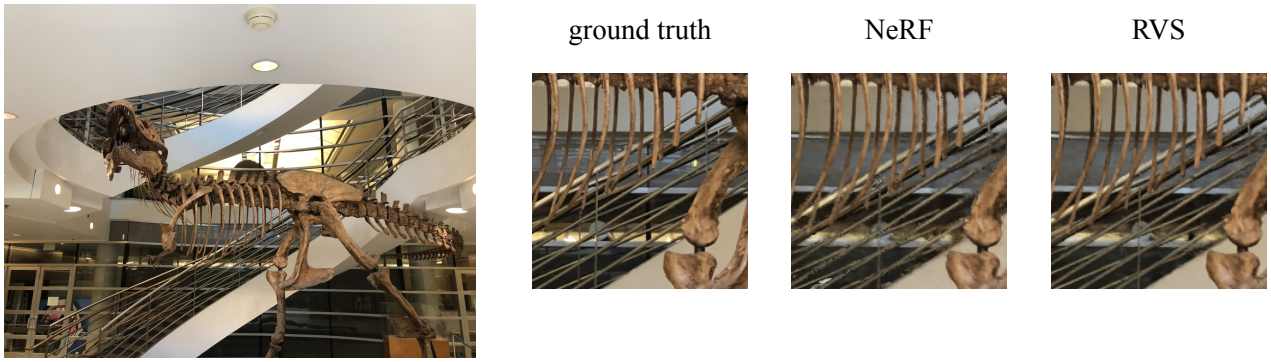

Figure 7: Comparison between renderings of a test-set view on T-Rex scene (LLFF) for $(32, 64)$ configuration. Artifacts can be seen on railings and bones in the NeRF render, while such artifacts are absent in the reconstruction produced by our model.

F PER-SCENE RESULTS

Table 10: Differentiable hierarchical sampling with NeRF on Blender dataset.

PSNR↑	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg
NeRF, $(N_p = 32, N_f = 64)$	31.33	23.89	28.26	35.44	30.11	28.65	31.46	26.76	29.49
RVS, $(N_p = 32, N_f = 64)$	31.99	24.60	29.27	36.18	31.89	29.31	31.84	27.19	30.26
SSIM↑	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg
NeRF, $(N_p = 32, N_f = 64)$	0.956	0.908	0.949	0.972	0.947	0.940	0.973	0.834	0.934
RVS, $(N_p = 32, N_f = 64)$	0.958	0.915	0.955	0.974	0.955	0.946	0.974	0.834	0.939
LPIPS↓	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg
NeRF, $(N_p = 32, N_f = 64)$	0.059	0.116	0.063	0.050	0.070	0.072	0.033	0.220	0.085
RVS, $(N_p = 32, N_f = 64)$	0.056	0.113	0.060	0.050	0.066	0.065	0.033	0.219	0.082

Table 11: Differentiable hierarchical sampling with NeRF on LLFF dataset.

PSNR↑	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Avg
NeRF, $(N_p = 32, N_f = 64)$	31.22	24.79	20.81	31.00	20.32	27.41	25.89	26.64	26.01
RVS, $(N_p = 32, N_f = 64)$	31.87	24.89	20.89	31.11	20.27	27.40	26.48	26.98	26.24
SSIM↑	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Avg
NeRF, $(N_p = 32, N_f = 64)$	0.938	0.771	0.678	0.875	0.630	0.822	0.858	0.798	0.796
RVS, $(N_p = 32, N_f = 64)$	0.942	0.773	0.681	0.878	0.630	0.823	0.869	0.795	0.799
LPIPS↓	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Avg
NeRF, $(N_p = 32, N_f = 64)$	0.203	0.309	0.328	0.187	0.338	0.226	0.279	0.310	0.273
RVS, $(N_p = 32, N_f = 64)$	0.193	0.310	0.328	0.182	0.340	0.223	0.267	0.311	0.270

Table 12: Differentiable hierarchical sampling with NeRF++ on LF dataset.

PSNR↑	Africa	Basket	Torch	Ship	Avg
NeRF++, $(N_p = 32, N_f = 64)$	26.36	21.38	23.72	24.53	23.99
RVS, $(N_p = 32, N_f = 64)$	27.31	21.58	24.60	25.01	24.63
SSIM↑	Africa	Basket	Torch	Ship	Avg
NeRF++, $(N_p = 32, N_f = 64)$	0.838	0.790	0.767	0.744	0.784
RVS, $(N_p = 32, N_f = 64)$	0.865	0.812	0.797	0.777	0.812
LPIPS↓	Africa	Basket	Torch	Ship	Avg
NeRF++, $(N_p = 32, N_f = 64)$	0.221	0.302	0.297	0.329	0.287
RVS, $(N_p = 32, N_f = 64)$	0.177	0.290	0.258	0.288	0.253

Table 13: Differentiable hierarchical sampling with NeRF++ on T&T dataset.

PSNR↑	Truck	Train	M60	Playground	Avg
NeRF++, $(N_p = 32, N_f = 64)$	21.18	17.16	16.96	21.55	19.21
RVS, $(N_p = 32, N_f = 64)$	21.62	17.32	17.48	22.05	19.62
SSIM↑	Truck	Train	M60	Playground	Avg
NeRF++, $(N_p = 32, N_f = 64)$	0.661	0.539	0.617	0.633	0.612
RVS, $(N_p = 32, N_f = 64)$	0.666	0.545	0.619	0.657	0.622
LPIPS↓	Truck	Train	M60	Playground	Avg
NeRF++, $(N_p = 32, N_f = 64)$	0.423	0.541	0.516	0.493	0.493
RVS, $(N_p = 32, N_f = 64)$	0.410	0.527	0.506	0.446	0.472

Table 14: DVGO with Monte Carlo estimates on Blender dataset.

PSNR↑	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg
DVGO	34.07	25.40	32.59	36.75	34.64	29.58	33.14	29.02	31.90
MC + RVS, $k = 4$	33.79	25.16	31.81	36.22	33.35	28.32	33.03	27.87	31.19
MC + RVS, $k = 8$	33.49	25.16	31.30	36.26	33.34	28.64	32.87	27.99	31.13
MC + RVS, adaptive k	34.13	25.18	31.17	36.63	33.85	29.09	33.03	28.43	31.44
SSIM↑	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg
DVGO	0.976	0.929	0.977	0.980	0.976	0.950	0.983	0.877	0.956
MC + RVS, $k = 4$	0.976	0.927	0.975	0.979	0.971	0.938	0.982	0.857	0.951
MC + RVS, $k = 8$	0.973	0.927	0.972	0.979	0.970	0.942	0.981	0.861	0.951
MC + RVS, adaptive k	0.977	0.928	0.972	0.980	0.973	0.946	0.982	0.870	0.953
LPIPS↓	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg
DVGO	0.028	0.080	0.025	0.034	0.027	0.058	0.018	0.162	0.054
MC + RVS, $k = 4$	0.028	0.079	0.028	0.038	0.032	0.070	0.017	0.181	0.059
MC + RVS, $k = 8$	0.031	0.080	0.030	0.037	0.033	0.067	0.019	0.178	0.059
MC + RVS, adaptive k	0.027	0.081	0.031	0.034	0.031	0.062	0.019	0.165	0.056