DDNeRF: Depth Distribution Neural Radiance Fields

David Dadon, Ohad Fried, Yacov Hel-Or
School of Computer Science, Reichman University, Herzliya, Israel
dadonda89@gmail.com , ofried@runi.ac.il, Toky@runi.ac.il

Abstract

The field of implicit neural representation has made significant progress. Models such as neural radiance fields (NeRF) [12], which uses relatively small neural networks, can represent high-quality scenes and achieve state-of-the-art results for novel view synthesis. Training these types of networks, however, is still computationally expensive and the model struggles with real life 360° scenes. In this work, we propose the ***depth distribution neural radiance field*** *(DDNeRF), a new method that significantly increases sampling efficiency along rays during training, while achieving superior results for a given sampling budget. DDNeRF achieves this performance by learning a more accurate representation of the density distribution along rays. More specifically, the proposed framework trains a coarse model to predict the internal distribution of the transparency of an input volume along each ray. This estimated distribution then guides the sampling procedure of the fine model. Our method allows using fewer samples during training while achieving better output quality with the same computational resources.*

1. Introduction

The field of implicit representation for 3D objects and scenes has been growing rapidly. Methods such as Occupancy Networks [10] and DeepSDF (Signed Distance Function) [15] achieved state-of-the-art results in 3D reconstruction. The two main advantages of implicit representation are compactness and continuity (compared to explicit representation methods such as meshes or voxels, which are discrete and less compact). It also enables reconstructing the 3D shape at any level of resolution by increasing/decreasing the number of samples in space. Due to their performance and accuracy, implicit methods became very popular and were adopted by many papers and in various domains.

Neural Radiance Fields (NeRF) [12] uses the same architecture as DeepSDF to represent a scene as a radiance field by answering the following query about a scene: Given a specific location (x, y, z) and a viewing direction (ϕ, θ), what is the RGB color and the density σ at this location and this viewing direction? When rendering an image, a pixel color (u, v) is evaluated by sampling points along the ray passing from the center of projection (COP) through the pixel (u, v), and applying a ray marching rendering technique for volume rendering [16]. NeRF achieved cutting-edge results for novel view synthesis, leading to what is called the "NeRF explosion". In the past two years, numerous follow-up works enhanced the original NeRF model and extended it with improved approaches and into new domains. We briefly review a few of these works in Section 2.

Nevertheless, NeRF has one major drawback: its extensive training and rendering time and space requirements. Because the quality of the model depends on the number of samples drawn along each ray (more samples produce better results), the training process has a trade-off between efficiency (number of samples) and quality. Most NeRF models use two-stage hierarchical sampling techniques. The first stage (coarse model) samples uniformly along each ray, dividing it into equi-length intervals. The opacity α and the total transparency of each interval i are used to determine the amount of influence w_i this interval has on the pixel's color (see Equation (2)). Each ray's w_i values are normalized and interpreted as a piecewise-constant PDF (or discrete PDF) between the intervals. The second stage (fine model) samples a new set of points along the ray according to this PDF function. Figure 1 (a) illustrates this method.

This paper proposes representing the coarse PDF as a mixture of Gaussian distributions rather than a piecewise-constant PDF. The input to the model is a specific ray interval and the outputs are parameters of a Gaussian distribution (μ, σ), representing the conditional density inside that interval (in addition to the color and total density). Theoretically, this representation of the internal density enables unlimited density resolution along the rays. It assists the sampling procedure of the second stage (fine model) by locating the fine samples around informative sections and investing most of the samples near the relevant phenomena. Figure 1 (b) illustrates our method. We will demonstrate and analyze its superiority over piecewise-constant PDF representation for a variety of domains and sampling budgets. Our model and

PDF representation are versatile and can be integrated into almost all existing NeRF models.
The main contributions of this paper are:

1. A continuous representation of the density distribution along rays in NeRF-based models, which leads to better results for a given number of samples. This allows NeRF models to be trained using fewer computational resources.

2. A novel distribution estimation (DE) loss, which provides an additional path for information to flow from the fine model backwards to the coarse model while improving the overall performance.

2. Related Works

2.1. Implicit 3D representation

The first two methods that achieved very good results using implicit representation of 3D objects were Occupancy Networks [10] and DeepSDF [15]. Occupancy Networks are trained to answer whether a point (x, y, z) in space is inside or outside a 3D object. DeepSDF is trained to predict the signed distance of a given point (x, y, z) from a 3D surface, where positive and negative distances represent a point outside or inside the shape, respectively. By answering the above queries for enough 3D points in a space, along with a variant of the marching cube algorithm, a 3D mesh of an object can be extracted. These methods became very popular due to their good results, compactness, and continuous representation. Additional works (such as Pifu [19]) were trained to answer more detailed queries to extract 3D shapes and textures. Advanced implicit models, e.g., SAL [2] and SALD [3], were developed to train models directly from raw 3D data without ground truth (GT).

2.2. The Implicit Representation of Radiance Fields

As described above, the NeRF [12] method uses a neural network to imply an implicit representation of a radiance field for volumetric rendering. It gets as an input an (x, y, z) location and viewing direction (ϕ, θ), and predicts as output the RGB color and the density α at this location. When this method was published, it achieved state-of-the-art results in the task of novel view synthesis. In the past two years, many works extended the NeRF model to additional tasks and domains. NeRF++ [24] extends the model to unbounded real world scenes using an additional neural network for background modeling and introducing new background parametrization. NeRF-W [9] extends the model for unconstrained image collections, and D-NeRF [17] extends it for dynamic scenes. Mip-NeRF [4] addresses the model aliasing problem with different resolution images. Several works also tried to reduce the required training time and, especially, the inference time [6, 8, 14, 18].

Two additional models that were recently published and achieved state-of-the-art results are plenoxels [20] and Mip-NeRF 360 [5]. Plenoxels use voxel grids with multiple small Multi-Layer Perceptrons (MLP) and significantly accelerate the training process. The amount of parameters Plenoxels use is considerably greater (about 100 times) than the regular NeRF model. The Mip-NeRF 360 model is much smaller than Plenoxels, but still uses about 10 times more parameters that the original Mip-NeRF model. Mip-NeRF 360 uses a novel architecture, space warping, online distillation, and regularization to significantly improve model performance on complex 360 degrees real world scenes.

The connection between sampling the rays around informative depth locations and the computational complexity appear in some of the above-mentioned works. DSNeRF [6] uses some prior depth information to improve the training time. It also allows training the model with a small number of images. NSVF [8] uses sparse voxel fields to achieve better sampling locations. DONeRF [14] improves inference time by using a depth oracle for sampling around informative locations. The depth oracle is trained with GT depth (or a trained NeRF model) to predict an accurate location for the second-stage sampling. DONeRF [14] also uses log-sampling and a space warping technique to increase the obtained quality on areas far away from the camera.

Our model can be seen as a direct extension of the original NeRF model [12] and Mip-NeRF [4]. The next two sub-sections provide a brief overview of these two models.

2.2.1 The Original NeRF Model

As mentioned above, NeRF receives a point and viewing direction as input (x, y, z, θ, ϕ), and produces a 4D output (R, G, B, σ), where R, G, B indicate colors and σ is the density of the input point along the viewing direction. Given an image whose intrinsic and extrinsic parameters are known, each pixel in that image defines a ray $\mathbf{r}$, initiated at the COP and passing through that pixel at point $\mathbf{o}$. To estimate the color of the pixel, the luminescent colors are accumulated along the ray: a set of n points are sampled along the ray $\mathbf{r}$ whose parameters (locations and viewing directions) are fed into the network, and their (R, G, B, σ) values are extracted as outputs. Assuming a length parameterization t along the ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{r}$, the sampled points are located at $\mathbf{r}(t_i),\ i = 1..n$, and (θ_i, ϕ_i) is determined by $\mathbf{r}$. These inputs are fed into the network whose outputs are $(R_i, G_i, B_i, \sigma_i)$. The density σ_i is translated into opacity α_i (having a value between 0 and 1) by considering the interval $\delta_i = t_{i+1} - t_i$ along the ray that is affected by σ_i:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i) \tag{1}$$

To avoid confusion with the σ in our model that represents standard deviation, we will omit σ in the notation from now

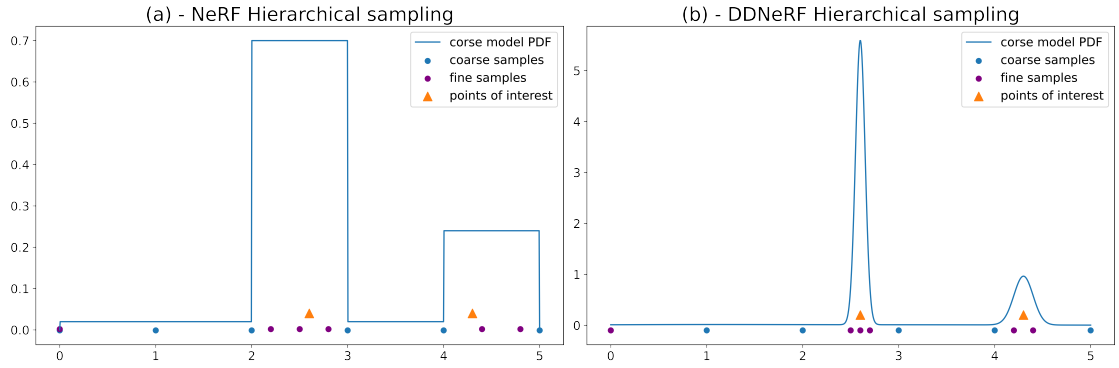

Figure 1. **Hierarchical sampling**: The horizontal axis represents the depth of the ray for a scene with a maximum depth of 5. First, the coarse samples (blue dots) are taken; then the density values are transformed into a PDF function (blue line). The fine model samples (purple dots) are taken with respect to the coarse model PDF. The orange rectangles represent points that have an influence on the pixel color (similar in both plots). The left plot illustrates the sampling procedure in the regular model; the right plot illustrates our scheme. Notice how the finer samples are concentrated around the informative areas in our representation.

on and refer directly to α. The influence of each interval i along the ray on the final color prediction is calculated as a combination of the accumulated transparency, $(1-\alpha_i)$, from interval i to the pixel, and the opacity values of interval i:

$$w_i = \alpha_i \cdot \prod_{j=0}^{i-1}(1-\alpha_j) \tag{2}$$

The NeRF architecture comprises two identical networks (coarse and fine), each with eight fully connected (FC) layers. The inputs are first encoded using positional encoding (PE) and then inserted into the network. As described in Section 1, the model uses two-stage hierarchical sampling where the w_i's values of the coarse model are normalized and can be interpreted as a discrete PDF h^c:

$$h_i^c = \frac{w_i}{\sum_{j=0}^{n-1} w_j} \tag{3}$$

where n is the number of samples along the ray. The fine model then samples the second stage of the hierarchical sampling according to h^c. Figure 1 (a) illustrates this process. Color rendering is performed using ray marching [16] for volumetric rendering and calculated as follows:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{n} w_i c_i \tag{4}$$

where $\hat{C}(\mathbf{r})$ is the predicted pixel color for ray $\mathbf{r}$, c_i is the RGB prediction for sample i and w_i is the influence that sample i has on the final RGB image. $\hat{C}_c(\mathbf{r})$ and $\hat{C}_f(\mathbf{r})$ are the coarse and fine model color predictions. The calculations of w_i and c_i are made separately for the coarse and fine models. The loss function is defined as:

$$L_{nerf} = \sum_{r\in\mathcal{R}} [||C(\mathbf{r}) - \hat{C}_f(\mathbf{r})||^2 + ||C(\mathbf{r}) - \hat{C}_c(\mathbf{r})||^2] \tag{5}$$

where $\mathcal{R}$ is the rays batch for loss calculation and $C(\mathbf{r})$ is the GT color for ray $\mathbf{r}$.

2.2.2 The Mip-NeRF Model

Mip-NeRF [4] is an extension of the regular NeRF model that was developed to handle aliasing artifacts when rendering images with different resolutions or at different distances than the images used in the training process. Instead of rays of lines, Mip-NeRF refers to a ray as a cone [1] whose vertex is at the COP and that passes through the relevant pixel with a radius related to the pixel size. The cone is divided into intervals (parts of the cone) along the depth axis and the network receives the encoding of an interval as an input. Each ray is divided into n intervals bounded by $n+1$ partitions $\{t_i\}$ where interval i is the cone volume bounded between partitions t_i and t_{i+1}. The bounded volumes are encoded using a novel integrated positional encoding (IPE) before being passed through the network. With this volumetric representation, the model can consider the entire volume that affects the pixel value. Mip-NeRF uses a single neural network for both the coarse and fine models. The rest of the process is similar to the original NeRF. In our work we also use the cone representation as in Mip-NeRF.

3. Problems with NeRF Models

When looking deeper into the NeRF hierarchical sampling strategy we observe two inherent issues. The first one is that for n samples, the depth resolution of the fine model cannot be better than $\frac{1}{n^2}$ of the scene depth. In other words, even if the coarse pass predicts that 100 percent of the samples should be placed in a single interval, the fine sampling will sample this single interval uniformly. To overcome this

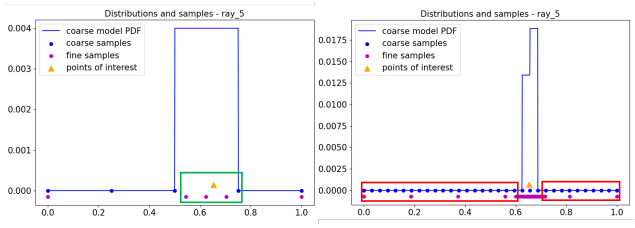

Figure 2. Left plot: Limited depth resolution when using a small number of samples (inside the green rectangle). Right plot: Many samples with zero contribution when using a large number of samples (inside the red rectangles).

problem we are forced to use a large number of samples during training. (A small number of samples will lead to inaccurate depth estimation.) Another derivative of this problem is that for a deep or unbounded scene, even when using a large number of samples, the model still struggles to achieve good results and there is a trade-off between background and foreground quality (as shown in NeRF++ [24]) as a function of the sample's depth range. Increasing the depth range will increase the background quality but concomitantly will decrease the foreground quality.

The second disadvantage we observed is that most samples in the coarse pass contribute almost nothing to the training process. Many samples predict zero influence from the very early training stages until its end. Despite this, we still need to use these samples along the entire training process in order to acquire dense enough samples in the second stage, due to the first problem mentioned above. Figure 2 illustrates the inherent trade-off between the two phenomena.

4. Method

4.1. Preliminaries

Our model is a direct extension of NeRF [12] and Mip-NeRF [4], described above in Sections 2.2.1 and 2.2.2. As mentioned, in the coarse step we are trying to extract additional information about the distribution of the density along the ray. We will show later that a better estimation of the density distribution along the ray in the coarse step will lead to improved localization of the fine samples, and subsequently improve the results.

To distinguish between coarse and fine samples, we denote $T^c = \{t_i^c\}_{i=1}^n$ as the coarse model samples and $T^f = \{t_i^f\}_{i=1}^n$ as the fine model samples. Similar to Mip-NeRF, our coarse model gets, as an input, an interval of a cone, but in addition to the regular $RGB\alpha$ output, our model also predicts an estimate of the density influence distribution inside this interval. More specifically, it predicts the mean μ and standard deviation (SD) σ of the distribution inside that interval. We assume the distribution inside each interval is Gaussian and that it neither affects nor is affected by adjacent intervals. The importance of this assumption will be clarified in Section 4.2.

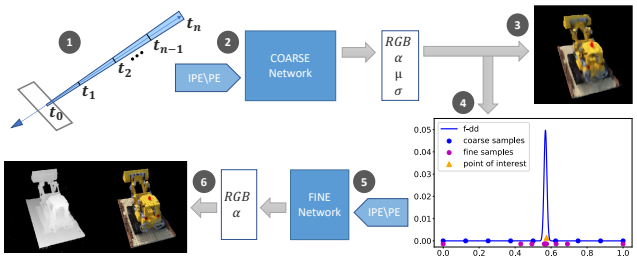

Figure 3. **DDNeRF full pipeline**: (1) Draw a cone in space and split it into relatively uniform intervals along the depth axis. (2) Pass these intervals through an IPE and then through the coarse network to get predictions. (3) Render the coarse RGB image. (4) Approximate the density distribution with respect to the coarse samples (blue dots) and their Gaussian parameters; then sample the fine samples (purple dots). (5) Pass these samples through an IPE and thereafter through the fine network to get predictions. (6) Render the final RGB image and depth map.

The coarse model learns to predict the distribution by trying to mimic the distribution of the fine model. We assume that the fine model achieves a better estimate of the density along the ray. This process will be described in more detail in Section 4.3. The entire pipeline of our model is shown in Figure 3. We used the Mip-NeRF [4] architecture with small modifications to add μ and σ to the coarse model predictions; see Section 1 in the Supplementary for more details.

4.2. Estimation of the Density Distribution

The predicted μ and σ are limited to be in the range between 0 and 1, by passing the predicted values through the sigmoid activation function. The values are interpreted relative to the interval length. The notation μ_i^r, σ_i^r stands for the relative mean and SD of interval i, respectively. The transformation from the relative interpretation to the absolute location and scale along the ray (μ_i, σ_i) is calculated as:

$$\mu_i = t_i^c + \mu_i^r \cdot \delta_i^c \quad ; \quad \sigma_i = \sigma_i^r \cdot \delta_i^c \tag{6}$$

where, $\delta_i^c = (t_{i+1}^c - t_i^c)$. These additional outputs enable us to achieve a finer distribution estimation along the ray. In addition to the discrete PDF estimation h^c between the intervals, we also estimate the distribution inside each interval. The total distribution along the ray is approximated as a linear sum of Gaussian distributions (one Gaussian for each interval). This representation allows us to focus the fine samples in a smaller and informative section along the ray.

The PDF inside interval i is denoted as $f_i(t) = \mathcal{N}(t|\mu_i, \sigma_i)$ and its CDF denoted as $F_i(t) = \int_{-\infty}^{t} f_i(\tau)d\tau$. Since we require that f_i be bounded inside the interval $\left[t_i^c, t_{i+1}^c\right]$, we truncate the Gaussian to be inside the interval

boundaries, normalize it, and define a truncated Gaussian distribution $f_i'(t),\ t \in \left[t_i^c, t_{i+1}^c\right]$:

$$f_i'(t) = \frac{1}{k_i} \cdot f_i(t)\ \ ;\ \ k_i = F_i(t_{i+1}^c) - F_i(t_i^c) \tag{7}$$

The truncation procedure is illustrated in Figure 4 (a). The normalized function $\{h_i^c\}$ between the intervals is calculated as it is done in the regular NeRF model (Equation (3)). The global density distribution estimated by the coarse model is then calculated as a mixture of the truncated Gaussians, where $\{h_i^c\}$ are used as the Gaussian weights:

$$f_{dd}(t) = h_i^c \cdot f_i'(t) \quad \text{for } \ t \in [t_i^c, t_{i+1}^c] \tag{8}$$

The entire procedure is illustrated in Figure 4.

The main reason we model the global distribution as a mixture of truncated Gaussians and not as a mixture of regular Gaussians is that we require each Gaussian to affect only a single interval. This property is necessary for two reasons. First, it enables calculation of each interval independently, while the results of one interval do not affect other intervals. Second, assigning each Gaussian to a specific interval allows efficient calculation of the samples in the fine model. It also enables efficient calculations of the loss component (described in Section 4.3) without requiring significant extra time or memory.

4.3. Training DDNeRF

During the training process we assume that the density distribution of the fine network is always more accurate than the coarse density distribution. Therefore, we promote the predicted coarse distribution to be close to the fine one.

The fine PDF function h^f is a discrete function computed similarly to Equation (3) with respect to density output α over the fine samples T^f. Our goal is to estimate h^f using the coarse model PDF function f_{dd}. We use the notation $\hat{h}^f$ to indicate the estimated h^f. Since f_{dd} is defined at each location along the ray, we can estimate $\hat{h}^f$ using f_{dd} and its CDF function F_{dd} as follows:

$$\hat{h}_i^f = Pr(t_i^f \le t \le t_{i+1}^f) = F_{dd}(t_{i+1}^f) - F_{dd}(t_i^f) \tag{9}$$

To train the network, we use the KL loss to define the divergence between the two discrete probabilities h^f and $\hat{h}^f$. Using the KL loss naively, however, tends to push μ and σ toward values close to 0 or 1 and impairs the model convergence (by over-shrinking the Gaussians or leading the model predictions to be in the vanishing gradient area of the sigmoid function). To avoid these issues, we add two regularization terms that encourage the Gaussian (before truncation) to remain in the center of the interval, with a SD large enough to avoid over-shrinking. This regularization also keeps the model inside the effective range of the sigmoid function. The regularization components of the loss function are $\sum_i \mu_{raw}^2$ and $\sum_i \sigma_{raw}^2$ where μ_{raw} and σ_{raw} are the model output values **before** passing through the sigmoid function to limit the range to be between 0 to 1. The overall DE loss is defined as follows:

$$DE_{Loss} = KL(\hat{h}^f, h^f) + \frac{1}{n} \cdot (\lambda_\mu \sum_i \mu_{raw}^2 + \lambda_\sigma \sum_i \sigma_{raw}^2) \tag{10}$$

where n is the number of coarse samples and λ_μ and λ_σ are the regularization coefficients. We set the coefficient values to be in the range of 0.01 to 0.1. The specific value depends on the number of samples along the rays. (The value for n samples is approximately $\frac{0.8}{n}$.); see Section 2 in the Supplementary for experimental results with different regularization values. We add the DE_{Loss} to the regular NeRF loss (eq. (5)) for our overall loss:

$$L = L_{nerf} + \lambda_{DE} \cdot DE_{Loss} \tag{11}$$

where λ_{DE} is the DE_{Loss} coefficient, set to be 0.1 in our experiments.

Smoothing and sampling: We define an uncertainty factor $u \ge 1$ that smooths the truncated f' Gaussians inside the intervals by increasing σ: $\hat{\sigma} = u \cdot \sigma$. This uncertainty factor is decreased during training toward 1 and corresponds to our increased certainty in the fine network estimation. Figure 5 describes the distributions and internal smoothing during training in our model vs. Mip-NeRF.

4.4. Dealing with unbounded scenes

For unbounded real world 360° scenes we tried an additional sampling strategy, which is similar to the log-sampling technique from DONeRF [14] for background areas. This extended model is denoted as DDNeRF* in Section 5. We also integrated our density distribution extension into the NeRF++ foreground model. This model is denoted as DDNeRF++ in the following experiments. See Section 1 in the Supplementary for additional details about these two extensions.

5. Experiments

We tested our model in three main domains: real-life forward-facing scenes, synthetic 360° scenes, and real life 360° scenes. We compare the performance of DDNeRF for different sampling budgets. We used the same number of samples for the coarse and fine networks. Thus, the number of samples listed in the results refers to one network. We used three different metrics when evaluating our results: structural similarity [23] (SSIM), perceptual similarity [25] (LPIPS) and PSNR.

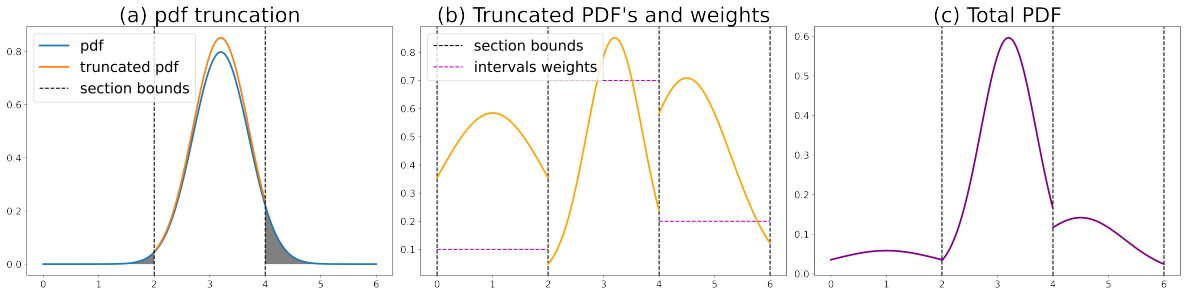

Figure 4. (a) PDF truncation process. The blue and orange curves are the PDF before (f_i) and after (f_i') truncation; gray marks the region outside the section boundaries. (b) Several adjacent truncated PDF distributions. Each orange truncated Gaussian is assigned to an interval; horizontal lines are interval weights h^c. (c) We combine all distributions into one distribution f_{dd}.

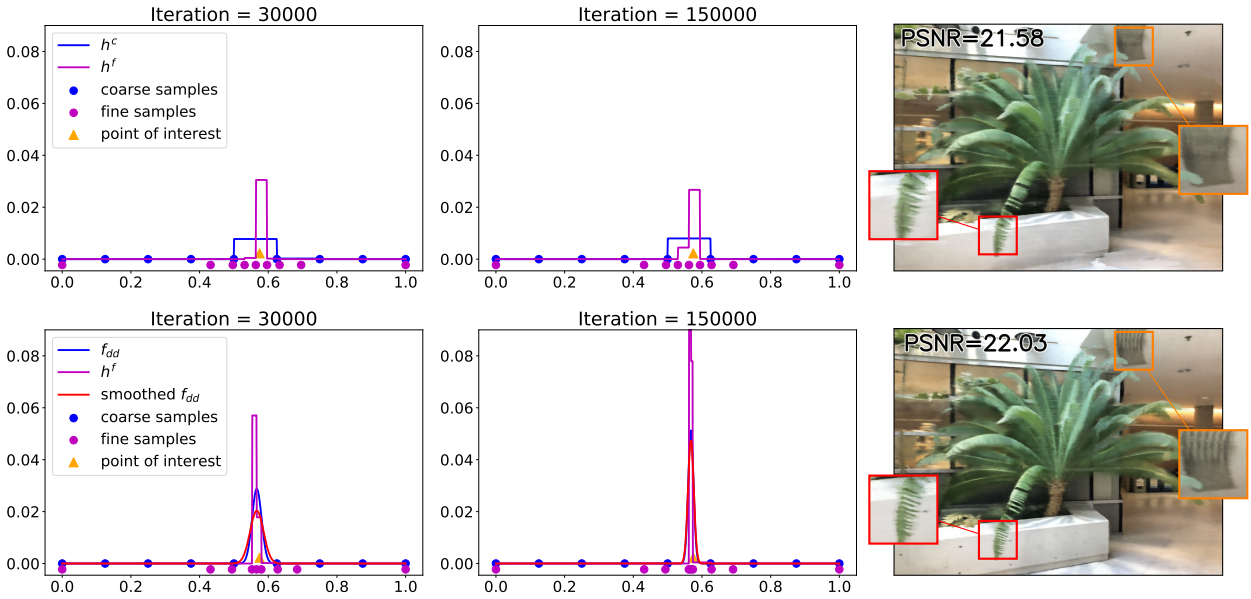

Figure 5. **Density distribution during training**. Training with **eight intervals**, Mip-NeRF is presented in the first row, DDNeRF in the second row. Left and middle columns: distribution along a chosen ray after 30k and 150k iterations, respectively. Right column: RGB image after 150k iterations. The blue curves are the coarse predicted PDF — h^c for Mip-NeRF and f_{dd} for DDNeRF. The red curves are the smoothed f_{dd}. Note how the divergence between red and blue is reduced during training as u decreases toward 1. Blue dots are the coarse samples, purple dots are the fine samples, and the purple curve is h^f. Our model (second row) keeps refining its fine samples' locations, while Mip-NeRF (first row) sample locations remain relatively static from 30K iteration until the end of the training process. Our model also achieves more accurate samples and a better RGB image compared to vanilla Mip-NeRF.

We divide our experiments into two parts. In Section 5.1 we focus on easy scenes in which NeRF achieved excellent results: real-life forward-facing and synthetic 360° scenes. Section 5.2 contains difficult scenes which are more challenging for NeRF: real life 360° bounded and unbounded.

We chose to compare our model with Mip-NeRF [4] (all scenes) and NeRF++ [24] (unbounded scenes). Plenoxels [20] achieves better results than the aforementioned models but uses many more parameters than our model (1-2 GB vs. 14 MB) so we chose to compare our model with models of the same order of magnitude. Section 2 in the Supplementary contains additional results and an ablation studies.

5.1. Part 1: Easy Scenes

For the forward-facing scenes we use the Fern and Trex scenes from LLFF [11]. We used the NDC (normalized device coordinates) transformation as in NeRF and Mip-NeRF. For the synthetic 360° scene we use the LEGO and the Ficus scenes from the NeRF [12] blender datasets. We train each model with 200K iterations using 2048 rays per iteration. To challenge the model we reduce the number of samples and repeat the training routine several times, with a different number of samples along the rays — 4, 8, 16, and 32. Validation is performed using the same number of samples as in the training. We compare our results with

Table 1. Results on the LLFF dataset (real-world forward-facing) and synthetic 360° scenes. We trained each model for 200k iterations. Our model achieved better results than the vanilla models for every number of samples. The values in the table are the mean values of each domain. For results per scene, see Section 2 in the Supplementary.

		LLFF			Synthetic		
Smpl	Model	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
4	Mip-NeRF	19.84	0.534	0.596	21.85	0.776	0.251
	DDNeRF	**20.40**	**0.561**	**0.573**	**21.93**	**0.777**	**0.237**
8	Mip-NeRF	21.43	0.638	0.453	24.50	0.845	0.173
	DDNeRF	**22.04**	**0.680**	**0.384**	**24.67**	**0.861**	**0.150**
16	Mip-NeRF	23.33	0.738	0.289	27.10	0.904	0.010
	DDNeRF	**23.61**	**0.759**	**0.262**	**27.66**	**0.922**	**0.0793**
32	Mip-NeRF	24.32	0.783	0.227	29.46	0.940	0.050
	DDNeRF	**24.39**	**0.789**	**0.219**	**30.17**	**0.951**	**0.039**

those of Mip-NeRF, which trained and validated the model results under the same conditions. Results are presented in Table 1. Our model achieves better results in each evaluation metric, regardless of the number of samples. Detailed per scene results as well as figures for qualitative comparisons are presented in Section 2 in the Supplementary.

5.2. Part 2: Difficult Scenes

5.2.1 Bounded scene:

For the bounded 360° scene we created our own scene of a motorcycle inside a warehouse. We acquired 200 snapshots from 360° views, where 10% of the images were saved for validation purposes. Although its depth is bounded, restoring this scene is not straightforward because it includes a big complex object and many small objects with fine details. We used the COLMAP structure from motion model [21, 22] to extract the relative orientation of the cameras. We trained each model with 300K iterations of 2048 rays per iteration. We varied the number of samples, using 32, 64, and 96 samples per ray. Results are shown in Table 2 and Figure 6. As can be seen from Table 2, our model achieves better results in all metrics for any number of samples. Moreover, our 32-sample model achieves better results than the 96-sample Mip-NeRF model. This is a significant reduction in memory and training time, despite the fact that for the same number of samples our model on average is 18% slower and uses 8% extra memory per iteration. The extra memory is needed for the f_{dd} calculations. Table 3 presents more details about the computation-quality trade-off.

Also, our model produces more accurate depth estimations and better RGB predictions, especially around complex shapes (Figure 6).

3D Mesh Extraction: We extract the 3D mesh by first extracting point-clouds from different views in the scene. Then, using open3D [26] we warp the main object points in the scene, remove outliers and use Poisson meshing to extract the object surface. DDNeRF reconstruction shows significantly better results compared to Mip-NeRF (Figure 7). Section 2 in the Supplementary offers for more results and details about the reconstruction process.

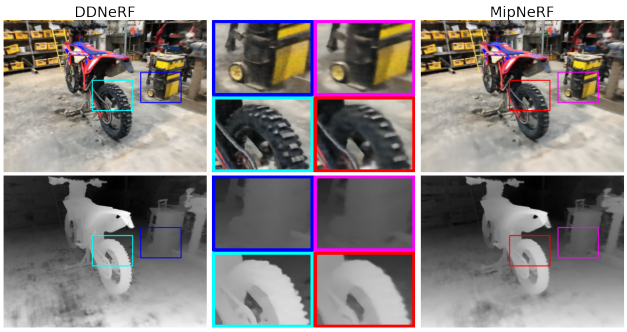

Figure 6. **Motorcycle scene results.** First row - RGB predictions. Second row - disparity estimation from the fine model results. Our model achieves better depth estimation and better RGB prediction. Zoom regions show how our model is able to represent complex shapes such as the motorcycle's off-road tires and the frame and small wheel of the tool cart.

Table 2. Results for real world 360° scenes. The Smpl column refer to samples in **each** of the networks (coarse and fine). The Sec/It column is a single iteration time in seconds and the Memory column is the maximum memory allocated in the GPU during training. All models were trained for 300K iterations. Performance was measured on an RTX3090 GPU.

Bounded scene – Motorcycle						
Smpl	Model	PSNR↑	SSIM↑	LPIPS↓	Sec/It	Memory
32	Mip-NeRF	20.36	0.533	0.532	0.053	1.73GB
	DDNeRF	**20.84**	**0.577**	**0.453**	0.066	1.84GB
64	Mip-NeRF	20.70	0.554	0.502	0.097	3.45GB
	DDNeRF	**21.07**	**0.592**	**0.422**	0.112	3.71GB
96	Mip-NeRF	20.80	0.563	0.488	0.142	5.17GB
	DDNeRF	**21.12**	**0.593**	**0.418**	0.163	5.67GB

Figure 7. **3D Reconstruction from point clouds**: The right object in each pair was reconstructed from DDNeRF, the left object was reconstructed from Mip-NeRF. DDNeRF achieves finer and less noisy shapes, especially around complex geometry regions.

Table 3. Computation–quality trade-off. We trained Mip-NeRF with 96 samples for 300k iterations on the motorcycle scene, and compared it to DDNeRF with different sample amounts and training iterations. We demonstrate the computational-quality trade-off using up to 300k iterations and 96 samples on DDNeRF. Green results are better than the baseline; red are inferior to the baseline. Smpl refers to samples in **each** of the networks (coarse and fine). The time column refers to the total training time in hours and the render column is inference time for a single 1008x756 RGB image. Our model achieves better results with a third of the samples or iterations. It can also render an image with better quality in approximately one third of Mip-NeRF's rendering time. Performance was measured on an RTX3090 GPU.

Model	Smpl	Iters	PSNR↑	SSIM↑	LPIPS↓	Time	Render
Mip-NeRF	96	300k	20.8	0.563	0.488	11.8H	20.91s
DDNeRF	32	300k	20.84	0.577	0.453	5.5H	**7.05s**
DDNeRF	96	80k	20.37	0.556	0.458	3.6H	21.37s
DDNeRF	96	100k	20.63	0.566	0.447	4.5H	21.37s
DDNeRF	96	140k	20.89	0.581	0.429	6.3H	21.37s
DDNeRF	96	300k	**21.12**	**0.593**	**0.418**	13.6H	21.37s

5.2.2 Unbounded scenes:

For unbounded scenes, we use the Playground and Truck scenes from the Tanks and Temples dataset [7]. We compare our models with both the Mip-NeRF and NeRF++ models. We train and test each model using 64 and 96 samples per ray. We also compare the DDNeRF* and DDNeRF++ methods that were described in Section 4.4. For NeRF++ and DDNeRF++ we split the sample budget equally between the foreground and background models. DDNeRF++ achieves the best scores on average, NeRF++ achieves a better PSNR score in the playground scene. When looking at the output images we can see that DDNeRF++ has better image quality than NeRF++ in both of the scenes in this domain. See Figure 8 and Table 4 for more details.

6. Conclusion:

In this paper we introduced DDNeRF, a general extension for NeRF models, which produces more accurate representations of the density along the rays while improving the sampling procedure and the overall accuracy of the results. We showed that our model produces superior results in various domains and sample amounts, while using fewer computational resources.

Our method is robust, self-supervised and can be adjusted to most NeRF models and enhance their performance. We successfully tested this with space warping and re-parametrization (NDC), and with different sampling methods (log-sampling). We implemented our extension on Mip-NeRF (general NeRF model) and NeRF++ (for general unbounded scenes) models. The models with our extension achieved better results in various domains.

Table 4. Results for real world 360° unbounded scenes. Smpl refer to samples in **each** of the networks (coarse and fine). All models were trained for 300K iterations.

Unbounded Scenes					
Smpl	Model	PSNR↑	SSIM↑	LPIPS↓	Scene
64	Mip-NeRF	21.47	0.547	0.540	Playground
	DDNeRF	21.71	0.568	**0.498**	
	NeRF++	**21.73**	**0.575**	0.524	
96	Mip-NeRF	21.67	0.551	0.550	
	DDNeRF	21.69	0.569	0.498	
	NeRF++	**22.19**	0.603	0.482	
	DDNeRF*	21.43	0.596	**0.451**	
	DDNeRF++	22.12	**0.611**	0.456	
96	NeRF++	21.43	0.646	0.411	Truck
	DDNeRF++	**21.52**	**0.655**	**0.377**	

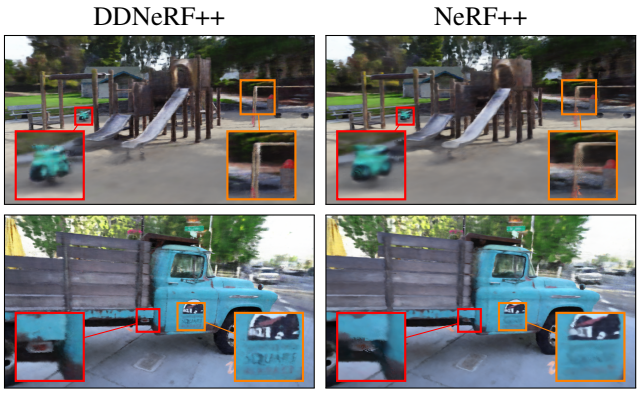

Figure 8. DDNeRF++ vs. NeRF++: Left images were rendered using the DDNeRF++ model, the right images using NeRF++. Notice that DDNeRF++ produces better results in foreground areas while background quality remain the same.

We expect our method to be integratable with any model which uses a two stage sampling strategy with the above properties (such as Mip-NeRF 360 [5]), and to improve their performance. However, integrating our method with Multi-resolution Hash Encoding [13] and Plenoxels [20] requires further examination and is a good direction for future research.

7. Acknowledgements:

This work was supported by the Israeli Ministry of Science and Technology under The National Foundation for Applied Science (MIA), and was partially supported by the Israel Science Foundation (grant No. 1574/21).

References

[1] John Amanatides. Ray tracing with cones. *SIGGRAPH Comput. Graph.*, 18(3):129–135, jan 1984.

[2] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[3] Matan Atzmon and Yaron Lipman. SALD: sign agnostic learning with derivatives. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[4] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICCV*, 2021.

[5] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022.

[6] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised nerf: Fewer views and faster training for free, 2021.

[7] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017.

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

[9] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*, 2021.

[10] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[11] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019.

[12] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[13] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.

[14] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum*, 40(4), 2021.

[15] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019.

[16] K. Perlin and E. M. Hoffert. Hypertexture. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, page 253–262, New York, NY, USA, 1989. Association for Computing Machinery.

[17] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *CoRR*, abs/2011.13961, 2020.

[18] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *CoRR*, abs/2103.13744, 2021.

[19] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization. *arXiv:1905.05172 [cs]*, May 2015. arXiv: 1905.05172.

[20] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022.

[21] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[22] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[23] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

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

[25] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[26] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.