Reichman University

Efi Arazi School of Computer Science

M.Sc. program - Research Track

DDNeRF: Depth Distribution Neural Radiance Fields

David Dadon

M.Sc. dissertation, submitted in partial fulfillment of the requirements
for the M.Sc. degree, research track, School of Computer Science
Reichman University (The Interdisciplinary Center, Herzliya)

June 2022

This work was carried out under the supervision of Prof. Yacov Hel-Or and Dr. Ohad Fried from the Efi Arazi School of Computer Science, Reichman University.

Abstract

In recent years, the field of implicit neural representation has progressed significantly. Models such as neural radiance fields (NeRF) [11], 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 very expensive. We present 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 by learning a more accurate representation of the density distribution along rays. More specifically, we train a coarse model to predict the internal distribution of the transparency of an input volume in addition to the volume's total density. This finer distribution then guides the sampling procedure of the fine model. This method allows us to use fewer samples during training while reducing computational resources.

Contents

Chapter 1

Introduction

The field of implicit representation for 3D objects and scenes has been growing rapidly in the last several years. Methods such as Occupancy Networks [9] and DeepSDF [14] (Signed Distance Function) have achieved state-of-the-art results in 3D reconstruction, which led to increased interest in this field. The two main advantages of implicit representation are compactness and continuity (compared to explicit representation methods such as meshes or voxels that are discrete and less compact). It also enables us to reconstruct the 3D shape for any level of detail (LOD) by increasing/decreasing the number of samples in space. Due to their performance and accuracy, implicit methods became very popular, adopted by many papers and various domains.

Neural Radiance Fields (NeRF) [11] use the same architecture as DeepSDF to represent a scene as a radiance field by answering the following query: given an (x,y,z) location and a viewing direction (ϕ,θ), what is the RGB color and the density σ in this location? When rendering an image, a pixel color is evaluated by sampling points along the ray that passes from the center of projection (COP) through the pixel, and applying a ray marching rendering technique for volume rendering [15].

At the time this method was published it achieved cutting-edge results for novel view synthesis. This led to what is called the “NeRF explosion”. In the past two years, numerous follow-up works improved the NeRF model and extended it to new domains. We briefly review a few of those works in the Chapter 2.

Nevertheless, NeRF has one major drawback: its extensive training 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 with respect to the depth axis along the ray and divides the ray into intervals according to these samples. The opacity (α) and the total transparency of each interval i are used to determine the amount of influence w_i of each interval on the pixel color (see Equation (2.2)). The w_i's values are normalized and interpreted as a piecewise-constant PDF (or discrete PDF) between the intervals. The second stage (fine model) samples points according to this PDF function. Fig. 1.1 (a) illustrates this method.

In this paper we propose to represent the PDF of the first hierarchical sampling stage as a mixture of Gaussian distributions and we will show its advantages over the piecewise-

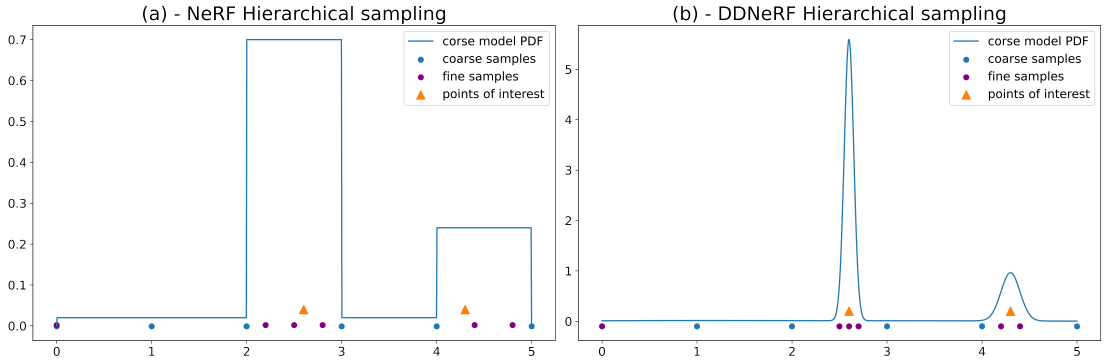


Figure 1.1: **Hierarchical sampling**: The horizontal axis represents the depth of the ray for a scene with max depth of 5. First, the coarse samples (blue dots) are taken; then the density values are transformed into a PDF function (blue line). The fine model samples (purple dots) are taken with respect 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.

constant PDF. The input to the model is a specific sub-section of the ray (interval), and the output is the Gaussian distribution parameters (μ, σ) of the density influence in that interval (in addition to the color and the total density). By using this technique, we achieve a more accurate density representation along the rays, which will allow us to receive more accurate samples in the second stage of the hierarchical sampling. Fig. 1.1 (b) illustrates our method. We will demonstrate and analyze its superiority over the piecewise-constant PDF representation for a variety of domains and sampling budgets. Our model and PDF representation are versatile and can be applied to almost each of the existing NeRF Models. **Our main contributions** can be described as follow:

1. A finer and more continuous representation of the density distribution along the ray in NeRF based models, which leads to better results for a given number of samples. This allows us to train the model with less computational resources.

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

Chapter 2

Related Works

2.1 Implicit 3D representation

Two of the first methods to achieve very good results for implicit representation of 3D objects were Occupancy Networks [9] and DeepSDF [14]. The Occupancy Networks [9] was trained to answer the query: "is this point inside or outside the 3D object?". The DeepSDF [14] was trained to answer the query: given an (x, y, z) location in space, what is the distance to the zero-level surface?, where positive and negative distances represent whether the point is located outside or inside the shape. By answering the above queries for enough 3D points in a space, combined with a variant of the marching cube algorithm, the 3D shape of an object can be extracted. These methods became very popular due to their good results, compactness and continuity characteristics. Additional papers (such as Pifu [18]) tried to answer similar queries to extract 3D shapes and textures. More advanced implicit models, e.g., SAL [2] and SALD [3], were developed to train directly from the raw 3D data without ground truth (GT).

2.2 NeRF models

As described above, the NeRF [11] method uses a neural network to imply implicit representation of radiance field for volumetric rendering. It gets as an input an (x, y, z) location and a view direction (ϕ, θ), and predicts as outputs the RGB color and the density α in 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++ [22] extends the model to unbounded-real world scenes using additional neural network for background modeling and new background parametrization. NeRF-W [8] extend the model for unconstrained image collection, and [16] extend it for dynamic scenes. MipNeRF [4] addresses the model aliasing problem with different resolution images. Many works also tried to reduce the required training time and, especially, the inference time [13] [5] [17] [7].
The connection between sampling around informative depth locations and the computational complexity appeared in some of the above-mentioned works. DSNeRF [5] uses some prior depth information to improve training time. It also allows training the model

with a small number of images. NSVF [7] uses sparse voxel fields to achieve better sampling locations. DONeRF [13] improves inference time by using a depth oracle for sampling in 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 [13] also uses log-sampling and space warping techniques to increase model quality on areas far from the camera.

Our model is a direct extension of NeRF [11] and MipNeRF [4]. The next two subsections are dedicated to describe those models in more detail.

2.2.1 NeRF

As mentioned above, NeRF receives a 5D input: (x, y, z, θ, ϕ), and produces a 4D output: (R, G, B, σ), where σ is the density of the input point that translates later into opacity α (value between 0 and 1) by considering the distance δ along the ray that is affected by σ. For sample i:

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i) \tag{2.1}$$

To avoid confusion with the σ in our model that represents standard deviation, we will omit σ in the notation from now on and refer directly to α. The influence of each sample i on the final color prediction w_i, is a combination of the accumulated transparency from sample i to the pixel and sample i opacity value (α_i):

$$w_i = \alpha_i \cdot \prod_{j=0}^{i-1} (1 - \alpha_j) \tag{2.2}$$

The NeRF architecture is composed of two identical networks (coarse and fine) with eight fully connected (FC) layers. The input is first encoded using positional encoding (PE) and then inserted into the network. As described in Chapter 1 (Introduction), 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:

$$h^c[i] = \frac{w_i}{\sum_{j=0}^{n} w_j} \tag{2.3}$$

Where n is the number of samples along the ray.
The fine model samples the second stage of the hierarchical sampling with respect to h^c. Fig. 1.1 (a) illustrates this process. Color rendering is performed using ray marching [15] for volumetric rendering and calculated as follow:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{n} w_i c_i \tag{2.4}$$

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 $\hat{C}_f(\mathbf{r})$

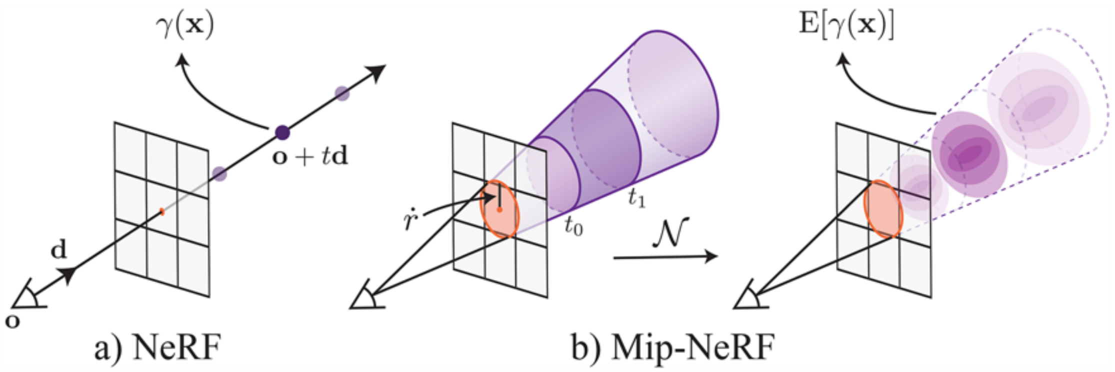


Figure 2.1: **NeRF vs mip-NeRF (image and caption was taken from mip-NeRF paper[4])**: “NeRF (a) samples points x along rays that are traced from the camera center of projection through each pixel, then encodes those points with a positional encoding (PE) γ to produce a feature $\gamma(x)$. Mip-NeRF (b) instead reasons about the 3D conical frustum defined by a camera pixel. These conical frustums are then featurized with our integrated positional encoding (IPE), which works by approximating the frustum with a multivariate Gaussian and then computing the (closed form) integral $E[\gamma(x)]$ over the positional encodings of the coordinates within the Gaussian.”

are the coarse and fine model color predictions. The calculations of w_i and c_i are made separately for the coarse and the fine models. The loss function is defined as:

$$L_{nerf} = \sum_{r\in R}[||C(\mathbf{r}) - \hat{C}_f(\mathbf{r})||^2 + ||C(\mathbf{r}) - \hat{C}_c(\mathbf{r})||^2] \tag{2.5}$$

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

2.2.2 MipNeRF

MipNeRF [4] is an extension of the regular NeRF model that was suggested to handle aliasing produced when rendering images at different resolutions or in different distances than the images used in the training process. Instead of a line, MipNeRF refers to a ray as a cone [1] with a vertex in the COP 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 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 the novel integrated positional encoding (IPE) method for volume encoding before being passed through the network. With this new ray representation the model prediction can encapsulate the entire volume that will affect the pixel value. Fig. 2.1 illustrates the main difference between NeRF and mipNeRF.

MipNeRF uses a single neural network for both the coarse and the fine models. The rest of the process is similar to the original NeRF. In our model we also use the idea that the model predicts information regarding the interval of a cone and not for a point on a ray.

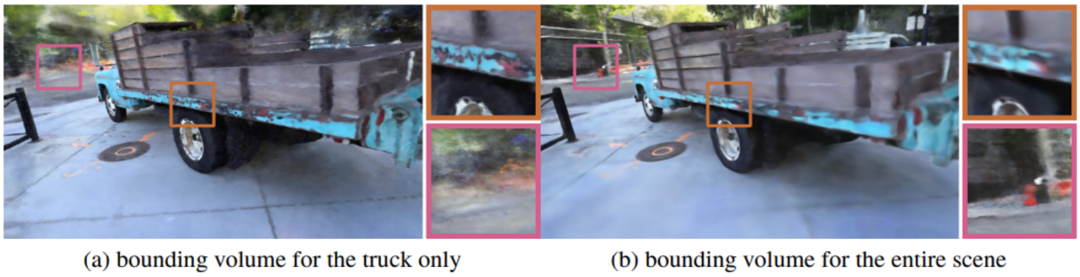
Chapter 3

Problem definition

When looking deeper into the NeRF hierarchical sampling strategy we observed two inherent disadvantages. The first one is that for n samples, the second pass sampling resolution cannot be better than $\frac{1}{n^2}$ of the scene depth. In other words, even if the first pass predicts that 100 percent of the samples of the second pass should be placed in a single interval, the finer sampling will sample this interval uniformly because the PDF along the ray is discrete.

To overcome this problem we are forced to use a large number of samples during training (a small number of samples will lead to a non-accurate 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 to foreground quality (as shown in Nerf++[22]) as a function of the sample's depth range. Fig. 3.1 illustrates this problem.

The second disadvantage we observed in the traditional NeRF sampling strategy is that most of the samples in the first pass contribute almost nothing to the training process because they predict zero influence from a very early stage in the training until its end. Despite this we still need to use them because of the first problem we mentioned. Fig. 3.2 illustrates the inherent trade-off between the two problems.


(a) bounding volume for the truck only

(b) bounding volume for the entire scene

Figure 3.1: **Foreground-Background Trade-off demonstration from NeRF++ paper [22]:** Scenes was rendered using regular NeRF model with the same number of samples. (a) The samples where taken in the foreground volume only. (b) The samples where taken in the entire scene volume. Foreground comparison: Note how the truck quality in the right image is lower due to a sparser samples around the truck location. Background comparison: Note how the background quality in the left image is lower because samples was taken with limited depth range.

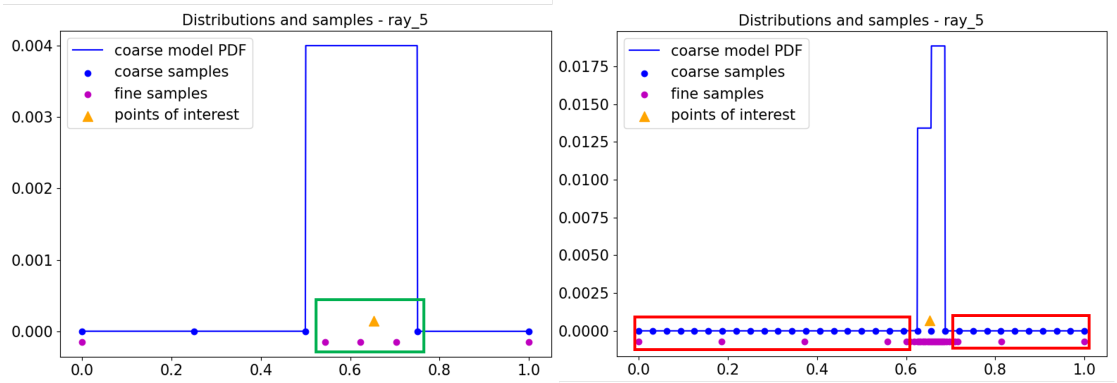


Figure 3.2: Left plot: Limited depth resolution when using a small number of samples (inside the green rectangle). Right plot: Many samples with zero contribution (inside the red rectangles).

Chapter 4

Method

4.1 Preliminaries

Our model is a direct extension of NeRF [11] and MipNeRF [4]. For this reason we described those models in more detail in section 2.2.1 and section 2.2.2.

4.2 General Description

As mentioned above, we are trying to extract additional information about the distribution of the density along the ray from our coarse model. We will show that a more accurate estimation of the influence distribution of the density along the ray predicted by the coarse network will lead to better fine samples and improved results.
To distinguish between the coarse and fine samples, we denote $T^c = \{t_i^c\}_{i=0}^n$ as the coarse model samples and $T^f = \{t_i^f\}_{i=0}^n$ as the fine model samples. Similar to MipNeRF, our coarse model gets, as an input, an interval of a cone, but in addition to the regular $RGB\alpha$ output, it also predicts an estimation of the density influence distribution inside this interval. More specifically, it predicts the mean μ and s.t.d. σ of the distribution inside that interval. We assume the distribution inside each interval is Gaussian and it does not affect or is affected by adjacent intervals. The importance of this assumption will be clarified later in this chapter.
The coarse network learns to predict the distribution by trying to mimic the fine network distribution. We assume that the fine network has a better estimation 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 Fig. 4.1.

4.2.1 Architecture:

We use the MipNeRF[4] architecture with two modifications: (1) We use two different networks for the coarse and fine models - similar to what was done in the original NeRF paper. (2) We change the final FC layer of the coarse model, adding μ and σ to the predictions. Fig. 4.2 describe our coarse network architecture.

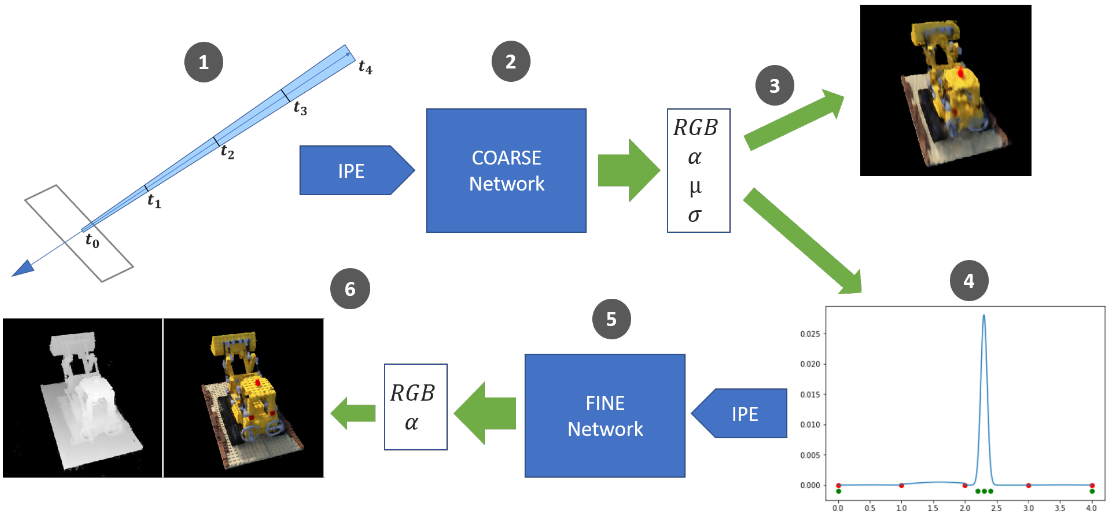


Figure 4.1: **DDNeRF full pipeline**: (1) Drawing a cone in space and splitting it into relatively uniform intervals along the depth axis. (2) Pass these intervals through an IPE and then through the coarse network to get predictions. (3) Render the coarse RGB image. (4) Approximate the density distribution and include the interval's internal distribution inside the coarse sections boundaries (red dots); then sample the fine samples (green 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.

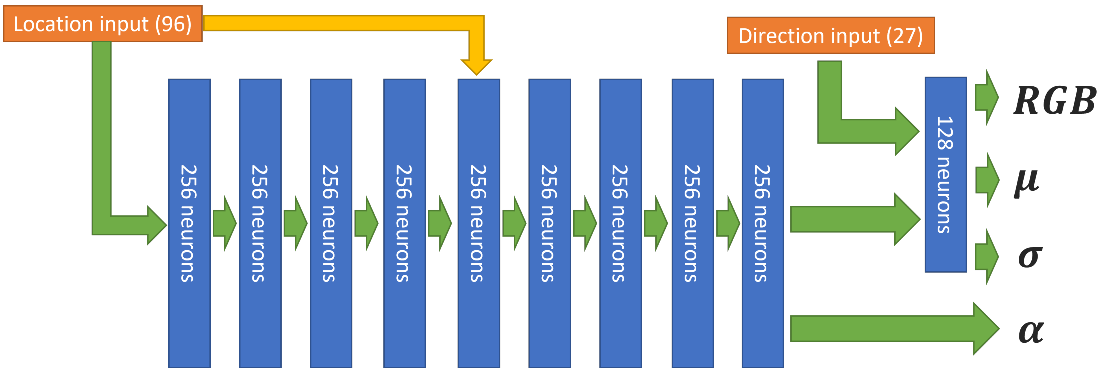


Figure 4.2: **DDNeRF Coarse network architecture**: The location input is after IPE. The direction input is after regular PE. Green arrows denote fully connected layers and the yellow arrow if for concatenation. Note that the density α is independent of the view direction while the other outputs depend on the view direction.

Similar to MipNeRF, we use IPE to encode the input sections before inserting them into the network. Although we chose to use the MipNeRF model, our model can be integrated with any variant of NeRF that uses hierarchical sampling by changing the coarse model.

4.3 Estimation of Density Distribution

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

$$\mu_i = t_i^c + \mu_i^r \cdot (t_{i+1}^c - t_i^c) \tag{4.1}$$

$$\sigma_i = \sigma_i^r \cdot (t_{i+1}^c - t_i^c) \tag{4.2}$$

These additional outputs allow us to achieve a finer distribution estimation along the ray. That means, that 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 combination of Gaussian distributions (one Gaussian for each interval) that allows us to focus the fine samples in a smaller area 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 $F_i(t) = \int_{-\infty}^{t} f_i(\tau)d\tau$. Because we want f_i to be bounded inside the interval, we need to truncate the Gaussian to be inside the interval boundaries, normalize it and define a truncated Gaussian distribution f'. The truncated Gaussian distribution f_i' inside interval i is defined as follows:

$$f_i'(t) = \frac{1}{k_i} \cdot f_i(t) \ \ ; \ \ k_i = F_i(t_{i+1}^c) - F_i(t_i^c) \tag{4.3}$$

The truncation procedure is illustrated in Fig. 4.3 (a). The discrete function h^c between the intervals is calculated as is done in the regular NeRF model (eq. (2.3)). The total density distribution estimated by the coarse model is a mixture of the truncated Gaussian models when h^c is used as the Gaussian weights. This distribution is denoted as f_{dd}. The calculation of f_{dd} is defined in eq. (4.4). The entire procedure illustrated in Fig. 4.3.

$$f_{dd}(t) = h^c[i] \cdot f_i'(t) \quad \text{when} \ \ t \in [t_i^c, t_{i+1}^c] \tag{4.4}$$

The main reason we prefer the mixture of truncated Gaussians over the regular mixture of Gaussians is that we want each Gaussian to affect only a single interval. This property is necessary for two reasons. First, because the model calculates each interval independently; we do not want the results of one interval to affect others. Second, assigning each Gaussian to one specific interval allows us to calculate the second pass samples and the additional loss component (described below) efficiently and without requiring significant extra time or memory.

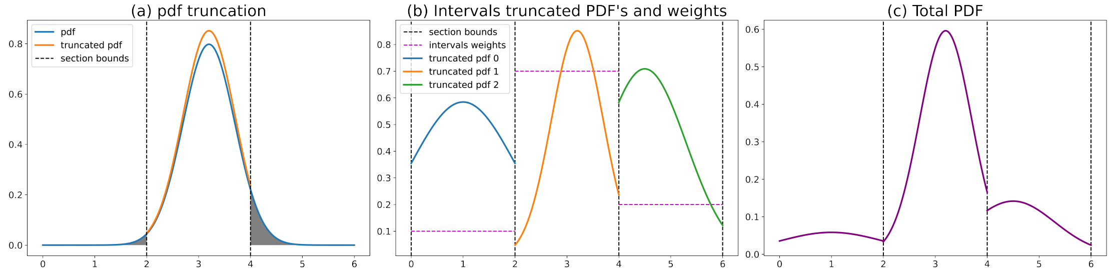


Figure 4.3: (a) The PDF truncation process. The tails that exceed the section boundaries are gray. The blue and orange curves are the PDF before (f_i) and after (f_i'), i.e., before and after the truncation. (b) Different intervals truncated PDF distributions and weights. Each color represents an interval. The vertical dashed lines are the interval bounds and the horizontal lines are the intervals weights (h^c). (c) The union of all distributions into one finer distribution f_{dd}.

4.4 Distribution Estimation Loss

To help the coarse network learn to approximate the density distribution, we assume that the density distribution of the fine network is always closer to the real density distribution. Hence we are forcing the predicted coarse distribution to be close to the fine one.

The fine PDF function h^f is a discrete function computed similarly to eq. (2.3) with respect to density output α of the fine samples T^f. We want to estimate h^f using the coarse model PDF function f_{dd}. We use $\hat{h}^f$ to denote the estimated h^f. Because f_{dd} is defined for every location on the ray, we can estimate $\hat{h}^f$ using f_{dd} and its CDF function F_{dd} as follows:

$$\hat{h}^f[i] = Pr(t_i^f \leq t \leq t_{i+1}^f) = \int_{t_i^f}^{t_{i+1}^f} f_{dd}(t)\,dt \; = F_{dd}(t_{i+1}^f) - F_{dd}(t_i^f) \tag{4.5}$$

We use KL divergence to measure the divergence between the two discrete probabilities h^f and $\hat{h}^f$.

Using the KL loss naively 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 the vanishing gradient area of the sigmoid function). To avoid these issues, we add two regularization terms encouraging the Gaussian (before truncation) to remain in the center of the interval, with s.t.d. 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 $\sum\limits_i \mu_{raw}^2$ and $\sum\limits_i \sigma_{raw}^2$ where μ_{raw} and σ_{raw} are the model outputs 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:

$$DE_{Loss} = KL(\hat{h}^f, h^f) + \frac{1}{n} \cdot (\lambda_\mu \sum_i \mu_{raw}^2 + \lambda_\sigma \sum_i \sigma_{raw}^2) \tag{4.6}$$

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 0.01 to 0.1. The specific value depends on the number of samples along the rays (the specific value for n samples is approximately $\frac{0.8}{n}$).

The DE_{Loss} is added to the regular NeRF loss (eq. (2.5)) so the overall loss is:

$$L = L_{nerf} + \lambda_{DE} \cdot DE_{Loss} \tag{4.7}$$

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

4.5 Sampling and Smoothing

Except for the unbounded scene case, the first sampling stage in our model is always sampled uniformly along the ray. As in MipNerf, we use a 2-tap max filter followed by a 2-tap blur filter for smoothing h^c before sampling the second stage. For a small number of samples (up to 16), the smoothness method became a simple 1D blur filter with $[0.1, 0.8, 0.1]$ values during training, which helps to achieve better accuracy in space (we found that this method works better for a small samples number in most scenes). For internal interval smoothing we defined an uncertainty factor $u \geq 1$ 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 it corresponds to our increased certainty in the fine network estimation. This strategy also helps our model to refine the second stage sample locations throughout the entire training process, while MipNeRF retains similar location from early stage in the training process. Fig. 4.4 visualizes the differences between the two sampling methods.

For an unbounded scene first pass, we also tried a different sampling strategy, a combination of principles from two methods. Similar to NeRF++[22], we dedicated half of the samples to uniformly sample the foreground volume. We do so by defining a maximum distance r_{max} from the origin (the scene center) that we considered as a foreground sphere. That sphere contains all camera locations in the scene, so when rendering an image, the maximum distance along the ray considered as foreground will be $\gamma = 2 \cdot r_{max}$ (See Fig. 4.5 for visual example). Outside the foreground range we used the DONeRF log-sampling method [13]. Using this method we can cover deep scenes more efficiently, by allocating more samples to the foreground area, then gradually decreasing the sampling rate in far areas. For n samples, the $\frac{n}{2}$ foreground uniform samples $t^{uniform}$ was taken according to eq. (4.8):

$$t_i^{uniform} = near \cdot (1 - \frac{i}{\frac{n}{2}}) + \gamma \cdot (\frac{i}{\frac{n}{2}}) \quad \text{when} \quad i \in [0, \frac{n}{2}] \tag{4.8}$$

While *near* is the rendered frustum near plane.

The background log-sampling $t^{log-samples}$ values is calculating using eq. (4.9):

$$t_i^{log-samples} = \gamma + (1 - \frac{\log(d_i - \gamma + 1)}{\log(far - \gamma + 1)}) \cdot (far - \gamma) \tag{4.9}$$

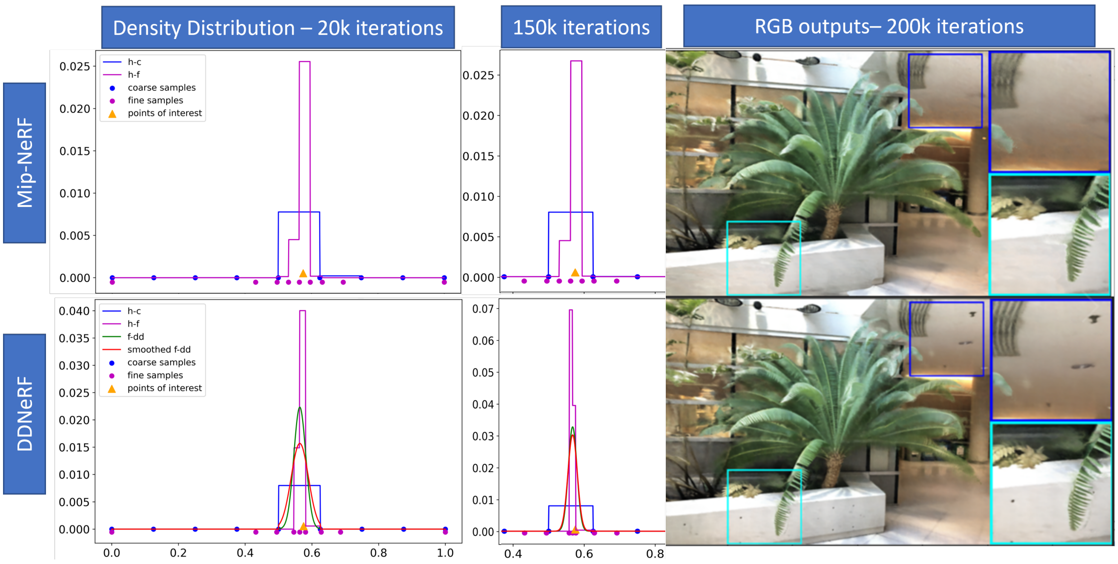


Figure 4.4: **Density distribution during training**. Training with eight intervals: 20k iterations (left), 150k iterations (middle), RGB image after 200k iterations (right). The blue line is h^c and the green line is f_{dd}. The red line is the smoothed f_{dd}; Note how the divergence between the red and green curves has closed during training as u decreased toward 1. The blue dots are the coarse samples. The purple dots are the fine samples and the purple line is h^f. Our model (second row) keeps refining its fine sample's locations, while the MipNeRF (firts row) sample's locations remain relatively similar from 20K to the end of the training process. Our model also achieves more accurate samples and a better RGB image than the regular mip-NeRF model.

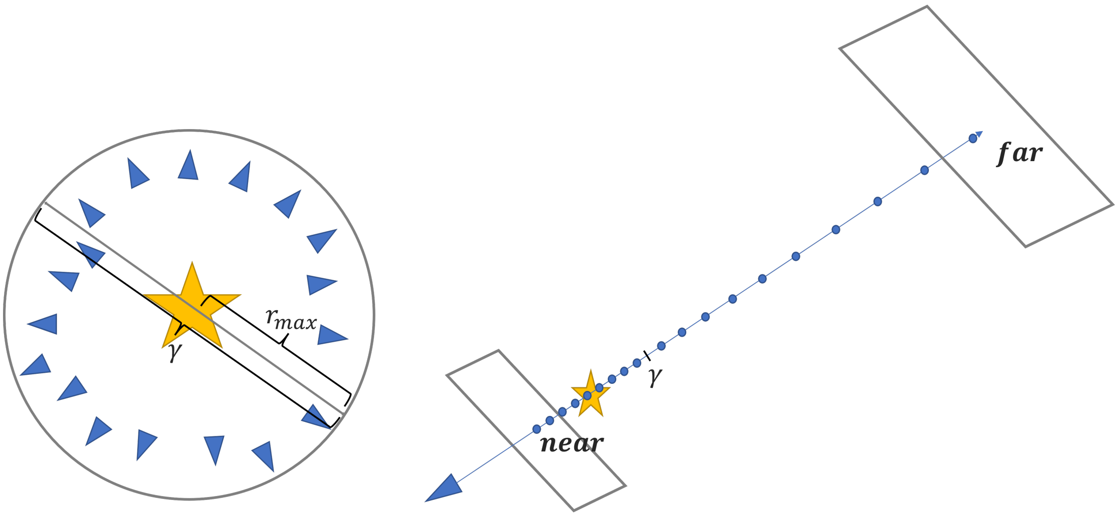


Figure 4.5: **Sampling strategy for unbounded scenes**: The blue rectangles represents camera locations. **Left part:** The yellow star is a foreground object located in the scene origin, the gray circle represent the foreground sphere that is used to calculate r_{max} and γ. **Right part:** The yellow star is a foreground object located inside the foreground range. The blue dots are the coarse samples (T^c) locations along the ray. Note the difference in the samples rate between the uniform sample from the near plane to γ and the log-samples from γ to the end of the ray (far plane).

Where far is the rendered frustum far plane and d_i values are calculated as follows:

$$d_i = \gamma \cdot (\frac{i}{\frac{n}{2}}) + far \cdot (1 - \frac{i}{\frac{n}{2}}) \quad \text{when} \;\; i \in [0, \frac{n}{2}] \tag{4.10}$$

Fig. 4.5 illustrated the first stage sampling strategy for unbounded scenes.
The second sampling stage remained the same.

This method performed better for this unbounded scene. Detailed results are described in Chapter 4.

Chapter 5

Experiments and Results

5.1 General description

We tested our model in three main domains: real-life forward facing scenes, synthetic 360° scenes and real life 360° scenes. We compare the model's performance 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 (SSIM), perceptual (LPIPS) and PSNR.
We divide our experiments into two parts. In part 1, we focus on domains in which NeRF achieved excellent results: real-life forward facing and synthetic 360°. Part 2 contains domains that NeRF is struggling with: real life 360° bounded and unbounded. All our training used a single GeForce GTX 1080.

5.1.1 Part 1:

For the forward facing scene we chose the fern scene from the LLFF paper [10]. We used the NDC transformation as in NeRF and MipNeRF. For the synthetic 360° scene we chose the LEGO scene from the NeRF [11] example datasets. We trained each model with 200K iterations using 2048 rays per iteration. To challenge the model we reduced the number of samples and repeated the training routine several times. Each time we used different numbers of samples along the rays – 4, 8, 16, 32. Validation was performed using the same number of samples as in the training. We compare our results with those of MipNeRF, which trained and validated the model results under the same conditions. Results are presented in Table 5.1 and in Fig. 5.1. Our model achieved better results in each one of the evaluation metrics for every number of samples.

Another indication of the additional information our model gathers relative to the other models is its depth estimation. We extract the depth as the mean of the PDF along the ray, for the regular coarse model – $E[h^c(t)]$ and for our coarse model – $E[f_{dd}(t)]$. For the fine models the depth calculated as $E[h^f(t)]$. Our coarse model produces a much better depth estimation than MipNeRF coarse model. Fig. 5.2 shows the qualitative results.

Table 5.1: Experiment results on the LLFF fern dataset (real-world forward facing) and synthetic 360 deg LEGO scene. We trained each model for 200k iterations. Our model achieved better results than the regular models for every number of samples.

		FERN			LEGO		
Samples	Model	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
4	MipNeRF	20.20	0.521	0.606	21.64	0.733	0.281
	DDNeRF	**20.81**	**0.577**	**0.507**	**21.79**	**0.741**	**0.274**
8	MipNeRF	21.6	0.614	0.477	24.64	0.813	0.184
	DDNeRF	**22.23**	**0.659**	**0.384**	**24.92**	**0.836**	**0.160**
16	MipNeRF	23.37	0.707	0.327	27.83	0.888	0.092
	DDNeRF	**23.51**	**0.727**	**0.285**	**28.67**	**0.917**	**0.062**
32	MipNeRF	23.85	0.740	0.279	30.38	0.932	0.045
	DDNeRF	**23.87**	**0.748**	**0.264**	**31.53**	**0.948**	**0.031**

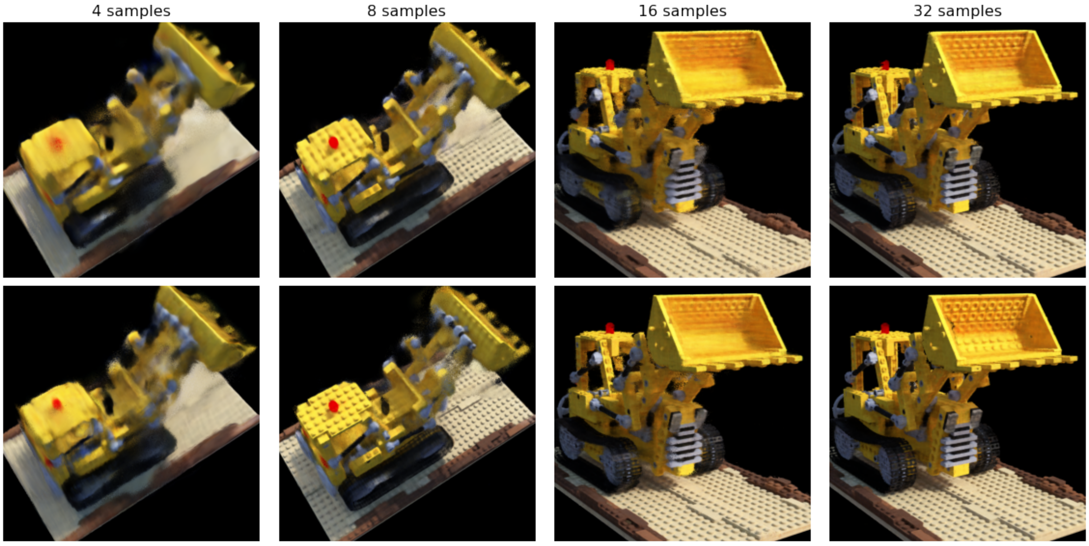


Figure 5.1: **Lego results**: First row – MipNeRF model. Second row – our model. The number of samples is marked at the top of each column. Our model achieves better results for any number of samples.

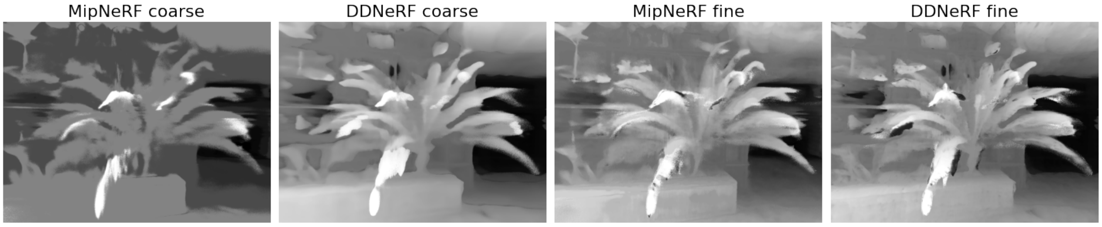


Figure 5.2: **Disparity comparisons:** Comparison between MipNeRF ans DDNeRF estimated disparity on the fern scene. Both models were trained and evaluated using eight samples and without NDC warping. Notice how the depth estimation of our coarse model is closer to the fine model's estimation than to the MipNeRF coarse model.

5.1.2 Part 2:

In the 360° domains we did not perform any space warping, excluding scale normalization of the world coordinate such that the main part of the scene is at a maximum distance of 1 from the origins.
For the bounded 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] [20] to extract the relative orientation of the cameras. We trained each model with 300K iterations using 2048 rays per iteration. As in the first part of our experiment, we used a different number of samples. In this case – 32, 64, 96. Results are shown in Table 5.2 and Fig. 5.3. As can seen from Table 5.2, our model achieved better results in all metrics for any number of samples. More than that, our 32 sample model achieved better results than the 96 sample MipNeRF model. Table 5.3 present more details about the computation-quality trade-off, which help to emphasize and analyze our model superiority. We can also see that our model produced more accurate depth estimation and better RGB prediction, especially around complex shapes (see Fig. 5.3).

In order to demonstrate the contribution for each of the components in our model we performed an ablation study. Result are presented in table 5.4.

For an unbounded scene we chose the playground scene from the Tanks and Temples dataset [6]. We compare our model with both the MipNeRF and NeRF++ models. We trained and tested each model using 64 and 96 samples. For NeRF++, we split the sample budget equally between the foreground and the background models. For the 96 samples we also compared the unbounded scene sampling method that we described in section 4.5. The model is notate as DDNeRF* in Table 5.2. Our model achieved the best LPIPS and SSIM scores from the models we tested. NeRF++ achieved a better PSNR score (see Table 5.2). When looking at the output images we can see that our model achieved better quality in the foreground and in the close background parts but struggles with far background parts; see Fig. 5.4

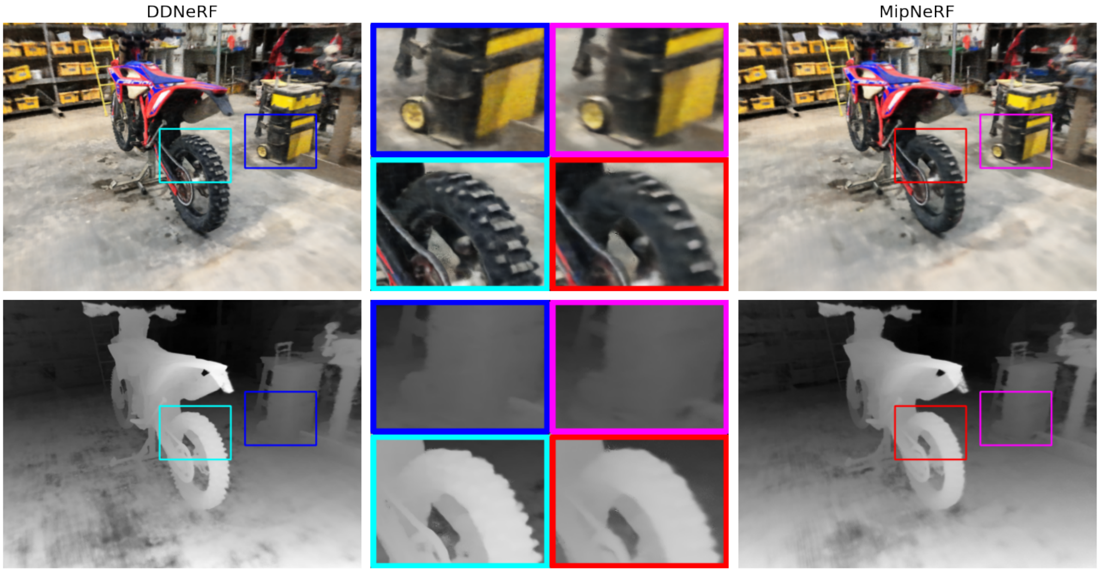


Figure 5.3: **Motorcycle scene results:** First row– RGB predictions. Second row– disparity estimation from the fine model results. Our model achieves better depth estimation and better RGB prediction. Notice the crop for how our model is able to catch complex shapes such as the motorcycle's off road tires and the frame and small wheel of the tool cart.

Table 5.2: Experiments results for real world 360deg scenes. "Smpl" column stands for the number of samples in each of the networks (coarse and fine). All models were trained for 300K iterations. The left part of the table compares DDNeRF with MipNeRF for different numbers of samples. The right part compare alse to NeRF++ model.

Bounded scene – Motorcycle					Unbounded scene – Playground				
Smpl	Model	PSNR↑	SSIM↑	LPIPS↓	Smpl	Model	PSNR↑	SSIM↑	LPIPS↓
32	MipNeRF	20.36	0.533	0.532	64	MipNeRF	21.47	0.547	0.540
	DDNeRF	**20.84**	**0.577**	**0.453**		DDNeRF	21.71	0.568	**0.498**
64	MipNeRF	20.7	0.554	0.502		NeRF++	**21.73**	**0.575**	0.524
	DDNeRF	**21.07**	**0.592**	**0.422**	96	MipNeRF	21.67	0.551	0.550
96	MipNeRF	20.8	0.563	0.488		DDNeRF	21.69	0.569	0.498
	DDNeRF	**21.12**	**0.593**	**0.418**		NeRF++	**21.74**	0.589	0.511
						DDNeRF*	21.43	**0.596**	**0.451**

Table 5.3: Computation-Quality trade-off. We trained mip-NeRF with 96 samples for 300k iterations on the motorcycle scene, and comparing it to DDNeRF with different sample numbers and training iterations. We demonstrate the computational-quality trade-off using up to 300k iterations and 96 samples on DDNeRF. Our model achieves better results with a $\frac{1}{3}$ of the samples or iterations. Green results are better than baseline, Red are inferior to baseline. x samples refers to x coarse samples and x fine samples.

Model	Samples	Iterations	PSNR↑	SSIM↑	LPIPS↓
MipNeRF	96	300k	20.8	0.563	0.488
DDNeRF	32	300k	20.84	0.577	0.453
DDNeRF	96	80k	20.37	0.556	0.458
DDNeRF	96	100k	20.63	0.566	0.447
DDNeRF	96	140k	20.89	0.581	0.429
DDNeRF	96	300k	**21.12**	**0.593**	**0.418**

Table 5.4: **Ablation study:** Motorcycle scene, 32 samples, 300k iterations. We train with constant variance value, without using uncertainty factor, using constant uncertainty factor, and finally our full model. The results show that predicting the variance is better than using constant variance for the learned distribution. The table is also demonstrates the importance of the uncertainty factor to the learning process.

Model	Version	PSNR↑	SSIM↑	LPIPS↓
MipNeRF	standard	20.36	0.532	0.532
DDNeRF	constant variance	20.62	0.567	0.462
DDNeRF	w/o uncertainty factor	20.39	0.555	0.453
DDNeRF	const uncertainty factor	20.67	0.571	**0.450**
DDNeRF	full model	**20.84**	**0.577**	0.452

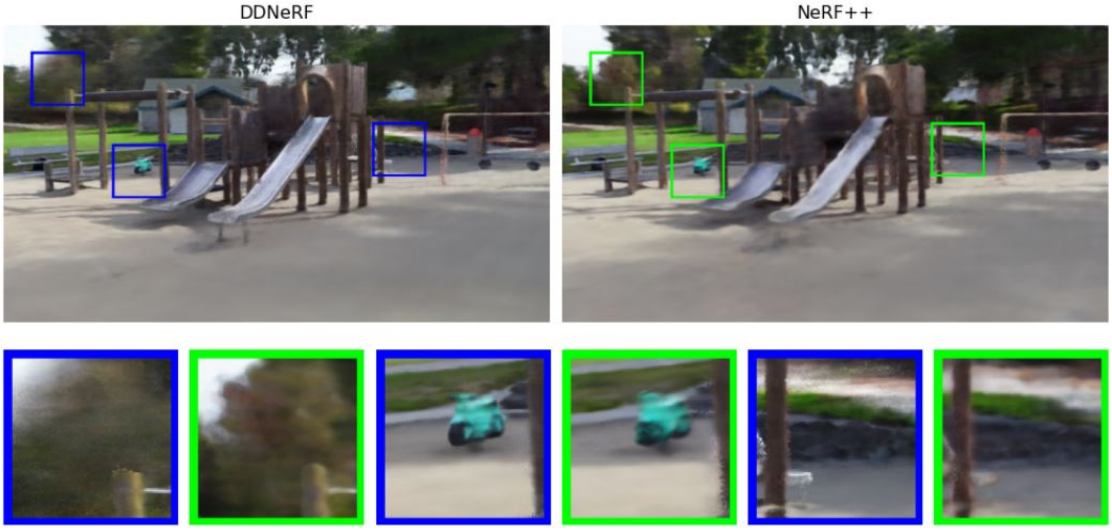


Figure 5.4: DDNeRF vs NeRF++: Left image was rendered using the DDNeRF* model, the right image using NeRF++. Notice that DDNeRF works better for the large foreground object. Two left crops: Far background – NeRF++ works better. Two middle crops: Small foreground objects – DDNeRF works better. Two right crops: Combination of foreground and close background – DDNeRF works better.

Chapter 6

Discussion and Conclusions

In this paper we introduced DDNeRF. An extension of the MipNeRF model that produces more accurate representation of the density along the rays while improving the sampling procedure and the overall results. We showed that our model provides superior results on various domains and sample numbers. DDNeRF uses fewer computational resources and produces better results.
Our method is robust, self-supervised and can be adjusted to most of the NeRF models to increase their performance. We successfully tested this assumption with:

1. Different space warping and reparameterization - NDC.

2. Different sampling method - log-sampling.

We expect our method to work better with any method who uses the same spatial representation. For example, methods that use OCTrees or multiple tiny NeRFs for reducing/parallel computations. We also expect our method to improve DONeRF performance. DONeRF uses depth GT or trained NeRF to train its oracle, producing better NeRF model could improve the oracle performance.
Due to the above properties we compared our model to a general NeRF model - mipNeRF and general unbounded scenes NeRF model - NeRF++. For unbounded scenes, despite the good results we got, we believe that combining our model for foreground and close background together with the NeRF++ [22] background model will lead to better results. We leave this for future work.
Integrating with two other recent works - Multi-resolution Hash Encoding [12] (SIGGRAPH 2022) and Plenoxels [19] (CVPR 2022) is less immediate and can also be a good future works.

Bibliography

[1] John Amanatides. “Ray Tracing with Cones”. In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 129–135. ISSN: 0097-8930. DOI: 10.1145/964965.808589. URL: https://doi.org/10.1145/964965.808589.

[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 et al. “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”. In: *ICCV* (2021).

[5] Kangle Deng et al. *Depth-supervised NeRF: Fewer Views and Faster Training for Free*. 2021. arXiv: 2107.02791 [cs.CV].

[6] Arno Knapitsch et al. “Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction”. In: *ACM Transactions on Graphics* 36.4 (2017).

[7] Lingjie Liu et al. “Neural Sparse Voxel Fields”. In: *NeurIPS* (2020).

[8] Ricardo Martin-Brualla et al. “NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections”. In: *CVPR*. 2021.

[9] Lars Mescheder et al. “Occupancy Networks: Learning 3D Reconstruction in Function Space”. In: *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[10] Ben Mildenhall et al. “Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines”. In: *ACM Transactions on Graphics (TOG)* (2019).

[11] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *ECCV*. 2020.

[12] Thomas Müller et al. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Trans. Graph.* 41.4 (July 2022), 102:1–102:15. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127.

[13] Thomas Neff et al. “DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks”. In: *Computer Graphics Forum* 40.4 (2021). ISSN: 1467-8659. DOI: 10.1111/cgf.14340. URL: https://doi.org/10.1111/cgf.14340.

[14] Jeong Joon Park et al. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. 2019. arXiv: `1901.05103 [cs.CV]`.

[15] K. Perlin and E. M. Hoffert. “Hypertexture”. In: *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’89. New York, NY, USA: Association for Computing Machinery, 1989, pp. 253–262. ISBN: 0897913124. DOI: `10.1145/74333.74359`. URL: `https://doi.org/10.1145/74333.74359`.

[16] Albert Pumarola et al. “D-NeRF: Neural Radiance Fields for Dynamic Scenes”. In: *CoRR* abs/2011.13961 (2020). arXiv: `2011.13961`. URL: `https://arxiv.org/abs/2011.13961`.

[17] Christian Reiser et al. “KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs”. In: *CoRR* abs/2103.13744 (2021). arXiv: `2103.13744`. URL: `https://arxiv.org/abs/2103.13744`.

[18] Shunsuke Saito et al. “PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization”. In: *arXiv:1905.05172 [cs]* (May 2015). arXiv: 1905.05172. URL: `http://arxiv.org/abs/1905.05172`.

[19] Sara Fridovich-Keil and Alex Yu et al. “Plenoxels: Radiance Fields without Neural Networks”. In: *CVPR*. 2022.

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

[21] Johannes Lutz Schönberger et al. “Pixelwise View Selection for Unstructured Multi-View Stereo”. In: *European Conference on Computer Vision (ECCV)*. 2016.

[22] Kai Zhang et al. “NeRF++: Analyzing and Improving Neural Radiance Fields”. In: *arXiv:2010.07492* (2020).

תקציר

ייצוג סתום באמצעות רשתות נירונים הוא תחום שהתפתח מאד בשנים האחרונות. מודלים כמו *NeRF* משתמשים ברשתות נירונים קטנות יחסית ומצליחים לייצג אובייקטים וסצנות תלת מימדיות באיכות גבוהה. עם זאת, אימון רשתות אלו עדיין דורש משאבי חישוב רבים.
בעבודה זו אנו מציגים את *Depth Distribution NeRF – DDNeRF* שיטה חדשה שמאפשרת לנו לייצג סצנות ואובייקטים תלת מימדיים בצורה מדוייקת יותר עבור כמות דגימות נתונה, להשיג תוצאות טובות יותר וע"י כך גם להקטין את משאבי החישוב הדרושים לאימון המודל. אנו משיגים זאת ע"י למידה מדוייקת יותר של התפלגות השפעת הצפיפות לאורך הקרניים המרונדרות על צבע הפיקסל. אנו מאמנים את המודל לחזות במהלך המעבר הראשון גם את ההתפלגות הפנימית של השקיפות בכל מקטע, בנוסף לצפיפות הכוללת של המקטע. באמצעות הייצוג הזה המעבר השני יכול לדגום במיקומים מדוייקים יותר לאורך הקרן ולהשיג תוצאות טובות יותר.

עבודה זו נכתבה בהנחיית פרופ' יעקב הל־אור וד"ר אוהד פריד, מבית הספר אפי ארזי למדעי המחשב, אוניברסיטת רייכמן

Reichman University Efi Arazi School of Computer Science

אוניברסיטת רייכמן

בית־ספר אפי ארזי למדעי המחשב

התכנית לתואר שני (M.Sc.)־ מסלול מחקרי

שיפור מודל *NeRF* ע"י למידה מדוייקת של התפלגות השפעת הצפיפות לאורך הקרניים

דוד דדון

עבודת תזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc. במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, אוניברסיטת רייכמן

יוני 2022