# DDNeRF: Depth Distribution Neural Radiance Fields — Supplementary

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

## 1. Additional Implementation Details

### 1.1. Architecture

We used the MipNeRF [1] architecture with two modifications. (1) We used two different networks for the coarse and fine models — similar to what was implemented in the original NeRF [3] paper. (2) We changed the last linear layer of the coarse model, adding $\mu$ and $\sigma$ to the predictions. Figure 1 describes our coarse network architecture. In Section 1.2.2 we discuss the integration of this model with NeRF++ [5].

Similar to MipNeRF, we used Integrated Positional Encoding (IPE) to encode the input intervals before inserting them into the network. The viewing direction was encoded using standard Positional Encoding (PE). Although our experiments are based of MipNeRF and NeRF++, our model can be integrated with any variant of NeRF that uses hierarchical sampling by changing the variant's coarse network.

### 1.2. Models For Unbounded Scenes

#### 1.2.1 Log Sampling Method

Both Mip-NeRF and the vanilla DDNeRF employ a uniform sampling strategy in the coarse network. For unbounded $360^\circ$ scenes, we also tried a different sampling strategy, a combination of principles from two methods. Similar to NeRF++, we dedicate half of the samples to uniformly sample the foreground volume. We implemented it by defining a maximum distance $r_{max}$ from the origin (the scene center) that we designate the foreground sphere. This sphere contains all camera locations in the scene. When rendering an image, the maximum distance along the ray considered as foreground is defined as $t_{max} = 2 \cdot r_{max}$ (see Figure 2 for a visual illustration). Outside of the foreground range we used the DONeRF [4] log-sampling method. Using this method we can cover more efficiently scenes with a large depth variations, because it enables to allocate more samples to the foreground space, then gradually reduce the sampling rate in distant areas. For $n$ samples, the $m = \frac{n}{2}$ foreground samples $t^{uniform}$ were taken as follows:

$$t_i^{uniform} = t_{near} \cdot (1 - \frac{i}{m}) + t_{max} \cdot (\frac{i}{m}) \text{ when } i \in \{0 \cdots m\} \quad (1)$$

where $t_{near}$ is the $t$ value of the near plane in the rendered frustum. The background log-sampling $t^{logSampl}$ values are calculated as follows:

$$t_i^{logSampl} = t_{max} + \left(1 - \frac{\log(d_i - t_{max} + 1)}{\log(t_{far} - t_{max} + 1)}\right) \cdot (t_{far} - t_{max}) \quad (2)$$

where $t_{far}$ is the $t$ value of the far plane in the rendered frustum, and the $d_i$ values are calculated as follows:

$$d_i = t_{max} \cdot (\frac{i}{m}) + t_{far} \cdot (1 - \frac{i}{m}) \text{ when } i \in \{0 \cdots m\} \quad (3)$$

Figure 2 illustrates the first-stage sampling strategy for unbounded scenes. The second sampling stage remaine the same.

This method, noted as DDNeRF*, achieves better performance than Mip-NeRF for unbounded scenes and its SSIM and LPIPS perceptual scores outperform the NeRF++ scores (See Table 4 in the main paper). Its main drawback is background modeling; NeRF++ produces better quality in the background areas.

#### 1.2.2 DDNeRF++

To overcome the background drawback, we integrated our model to the NeRF++ foreground model, i.e. we replaced the NeRF++ foreground model with DDNeRF. The background model remains similar as in NeRF++. We denote this new model – DDNeRF++. The foreground model is trained similarly to DDNeRF, and the main difference is that when calculating the $DE_{loss}$ component we consider only rays that intersect with the foreground areas. DDNeRF++ achieves better foreground quality then NeRF++ while maintaining the same background quality. Figure 3 demonstrates the quality difference between the methods.

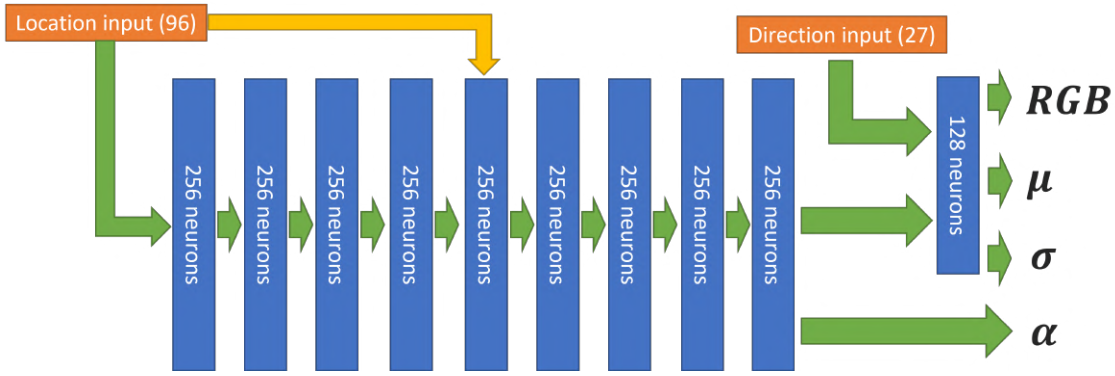


Figure 1. **DDNeRF Coarse network architecture**: The location input is used the Integrated Positional Encoding (IPE), while the direction input is used the regular Positional Encoding (PE). Green arrows indicate fully connected layers and the yellow arrow indicates concatenation. Note that the density $\alpha$ is independent of the viewing direction while the other outputs depend on the viewing direction.

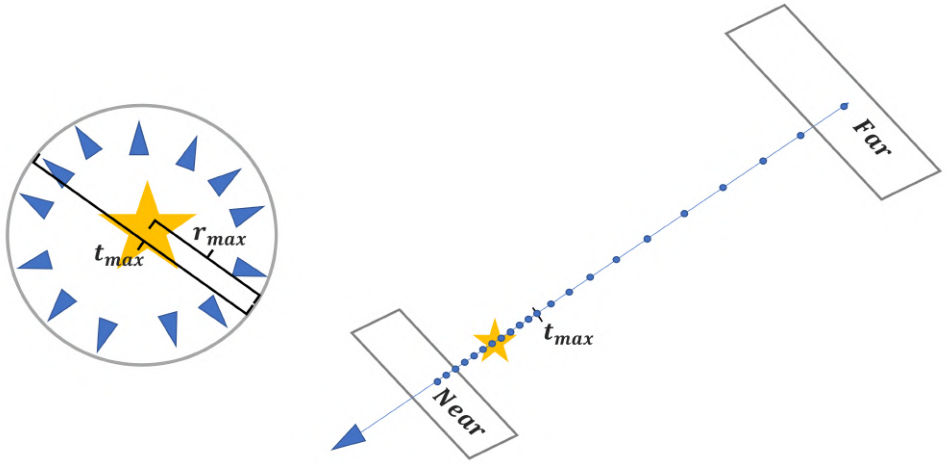


Figure 2. **Sampling strategy for unbounded scenes**: The blue triangles represent camera locations. **Left:** The yellow star indicates a foreground object located at the origin; the gray circle represents the foreground sphere defined by $r_{max}$ and $t_{max}$. **Right:** The yellow star is a foreground object located inside the foreground range. The blue dots are the coarse samples, ($T^c$), along the ray. Note, the difference in the sampling rate between the section from the near plane to $t_{max}$ (uniform sampling) and the the section from $t_{max}$ to the far plane (log sampling).

### 1.3. Smoothing

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) we used a simple 1D blur filter with $[0.1, 0.8, 0.1]$ values. We have found that this smoothing works better for a small number of samples in most scenes. For smoothing the internal interval we used the uncertainty factor $u$ as described in the main paper.

## 2. Additional experiments

This Section presents further qualitative and quantitative results and comparisons on the tested datasets. It also provides ablation study, evaluation of the predicted distribution quality, analysis for model size, and performances and analysis of the regularization terms.

### 2.1. Additional Results

Table 1 shows quantitative results per scene for the forward-facing and synthetic domains. Figures 4 and 5 illustrates the qualitative superiority of our model over MipNeRF on synthetic and forward-facing scenes for different amount of samples.

Figure 6 demonstrate the superiority of our model for $360^\circ$ scenes. Notice, that our model achieves better results with fewer samples.

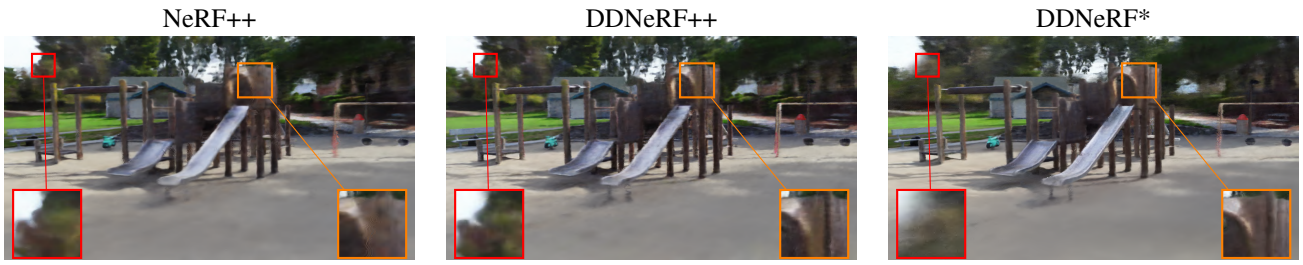


Figure 3. Unbounded scenes comparisons: Left images were rendered using the NeRF++ model, the middle image using DDNeRF++ and the right image using DDNeRF*. Relatively to NeRF++, DDNeRF* improves the foreground performance but struggle with background modeling. Notice that DDNeRF++ is manage to produces better results in foreground areas while background quality remain the same.

Table 1. Experiment results on the LLFF fern and trex scenes, (real-world forward-facing) and synthetic 360° LEGO and Ficus scenes. We trained each model for 200k iterations. Our model achieved better results than the regular models.

| | | FERN | | | TREX | | | LEGO | | | Ficus | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Smpl | Model | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| 4 | MipNeRF | 20.20 | 0.521 | 0.606 | 19.48 | 0.548 | **0.585** | 21.64 | 0.733 | 0.281 | **22.07** | **0.819** | 0.221 |
| | DDNeRF | **20.81** | **0.577** | **0.507** | **20.23** | **0.565** | 0.598 | **21.79** | **0.741** | **0.274** | **22.07** | 0812 | **0.199** |
| 8 | MipNeRF | 21.60 | 0.614 | 0.477 | 21.26 | 0.662 | 0.429 | 24.64 | 0.813 | 0.184 | 24.37 | 0.878 | 0.163 |
| | DDNeRF | **22.23** | **0.669** | **0.384** | **21.85** | **0.701** | **0.384** | **24.92** | **0.836** | **0.160** | **24.42** | **0.886** | **0.140** |
| 16 | MipNeRF | 23.37 | 0.707 | 0.327 | 23.30 | 0.770 | 0.250 | 27.83 | 0.888 | 0.092 | 26.38 | 0.920 | 0.108 |
| | DDNeRF | **23.51** | **0.727** | **0.285** | **23.71** | **0.791** | **0.239** | **28.67** | **0.917** | **0.062** | **26.66** | **0.927** | **0.096** |
| 32 | MipNeRF | 23.85 | 0.740 | 0.279 | 24.79 | **0.826** | 0.175 | 30.38 | 0.932 | 0.045 | 28.54 | 0.949 | 0.054 |
| | DDNeRF | **23.87** | **0.748** | **0.264** | **24.91** | **0.829** | **0.175** | **31.53** | **0.948** | **0.031** | **28.81** | **0.954** | **0.047** |

#### 2.1.1 3D Reconstruction

Using Marching cubes to extract 3D shape from NeRF is performed by sampling points inside a 3D voxel grid and using those points as inputs to a NeRF Model. This procedure is not perfectly aligned with our model that gets as an input an interval of a cone. We believe this issue is beyond the scope of this paper and we extract 3D mesh from point clouds that we extracted from different views of the scene. The full procedure is performed the follows:

1. To avoid smash artifacts on the edges of the object (from specific views), we filter points with normals that are close to be perpendicular to the viewing direction.

2. We define a cube that contains the main object and remove all points that are outside that cube.

3. We remove outliers and cluster points using the DBSCAN [2] algorithm.

4. We take the inlier points and use Poisson meshing to extract the object surface.

Using the above method as an alternative to the marching cubes, we also take into account floating artifacts that often appear in complex scenes. Those artifacts appear as small floating dense areas in the generated scene.

Although the model density prediction does not depend on the viewing direction, the point cloud is calculated as $\mathbb{E}[h^f(t)]$ from a specific viewing location. Thus, point clouds extracted from different viewing directions may not be aligned perfectly one on top of the other. Since the object geometry from a specific viewpoint is not represented as a binary surface but as $\mathbb{E}[h^f(t)]$, it is possible that the expectation of two different rays looking at the same point, but passing through different volumes, will be different. This is due to different densities along the rays or occlusion artifacts. See Figure 8 for intuitive illustration of this effect. Using our proposed method, instead of marching cubes, may be less accurate, but it represents the 3D perception we obtain when rendering a novel view. 3D reconstruction using Mip-NeRF and DDNeRF are presented in Figure 7. DDNeRF shows significantly better results.

### 2.2. Ablation study

To demonstrate the contribution of each component in our model we performed an ablation study. Table 2 demonstrates the importance of the uncertainty factor and the learned variance (instead of defining a constant variance). Figure 9 demonstrates the importance of the regularization component in the loss function and illustrates its effect on the Gaussian's parameter values.


Figure 4. **Forward-facing scenes results.** Each column was generated using different number of samples. The first row in each scene was generated by DDNeRF while the second row by Mip-NeRF. Our model produces better quality images for both scenes and for each number of samples.

### 2.3. Lighter coarse model

We perform additional study in order to examine the possibility to reduce the number of parameters in the coarse model. We examined the effect of the coarse model's size on the model performance. We tested two scenarios: In the first scenario we reduced the number of parameters in the coarse model and kept the fine model with the same number of parameters. In the second scenario, we reduced the number of parameters in the coarse model and added parameters to the fine model, while keeping the total number of parameters constant. Results are presented in Table 3. It is demonstrated that reducing the size of the coarse model causes a negligible decrease in quality while improving the model performance in both time (up to 27% faster) and space (up to 30%). On the other hand, when adding those parameters to the fine model, the overall model quality increases while

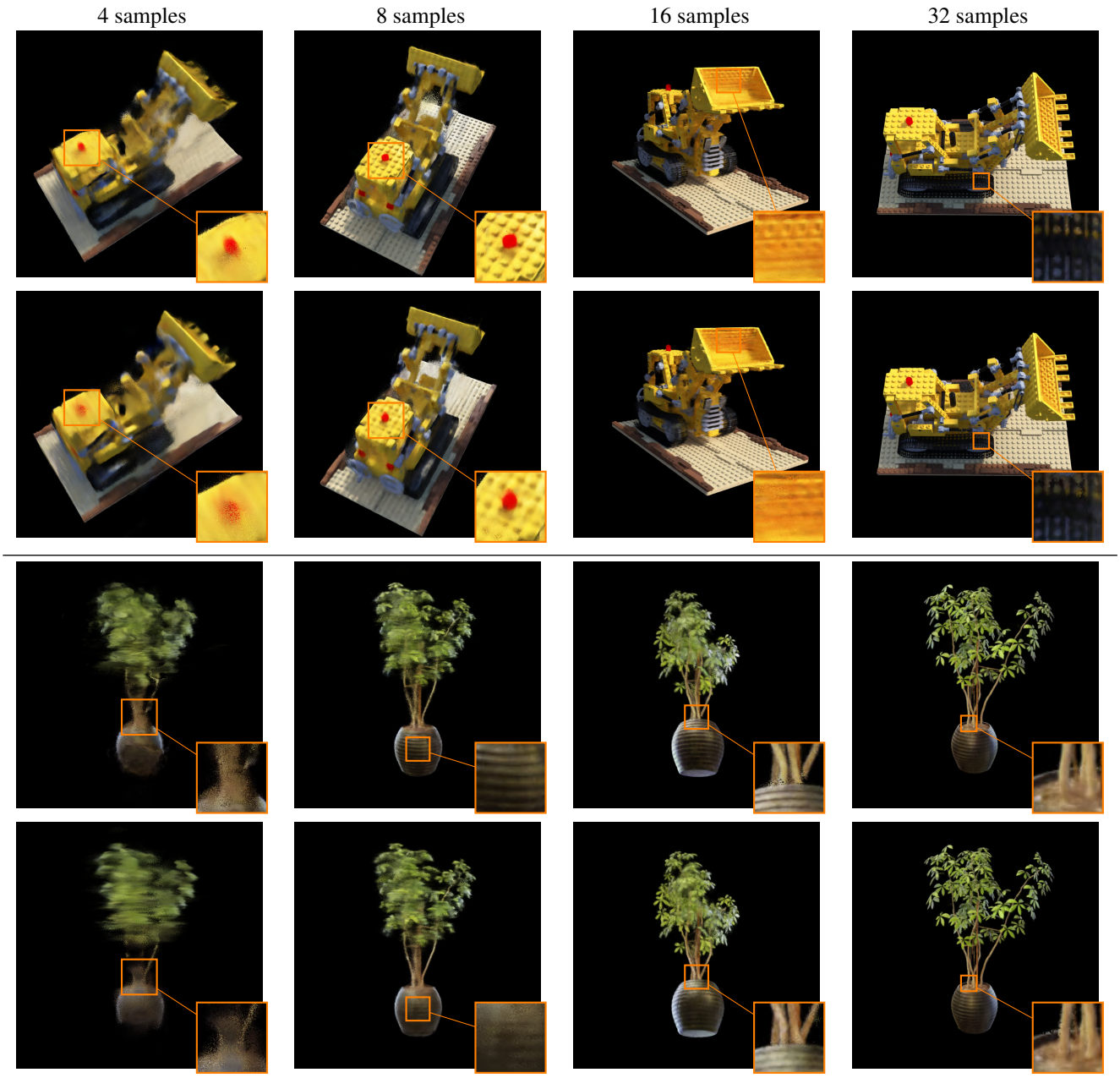


Figure 5. **Synthetic scenes results.** Each column was generated using different number of samples. The first row in each scene was generated by DDNeRF while the second row by Mip-NeRF. Our model produces better quality images for both scenes and for each number of samples.

time and space remain relatively the same.

### 2.4. DDNeRF Distribution Evaluation

To evaluated our predicted distribution in the coarse model, we tested this distribution on the Fern scene from the LLFF dataset. We extracted the depth as the mean of the pdf along the ray. We compare mipNeRF's coarse model, $\mathbb{E}[h^c(t)]$, with our coarse model, $\mathbb{E}[f_{dd}(t)]$. For the fine models, the depth is calculated as $\mathbb{E}[h^f(t)]$. Our coarse model produces a much better depth estimation than the MipNeRF coarse model. Figure 10 shows qualitative results.

For quantitative evaluation, we took two different views of that scene, extract key-points, calculate the fundamental matrix between those images (to remove outliers), and estimate inlier 3D points using rays intersection. We refer those 3D points as our GT values for the evaluation process.

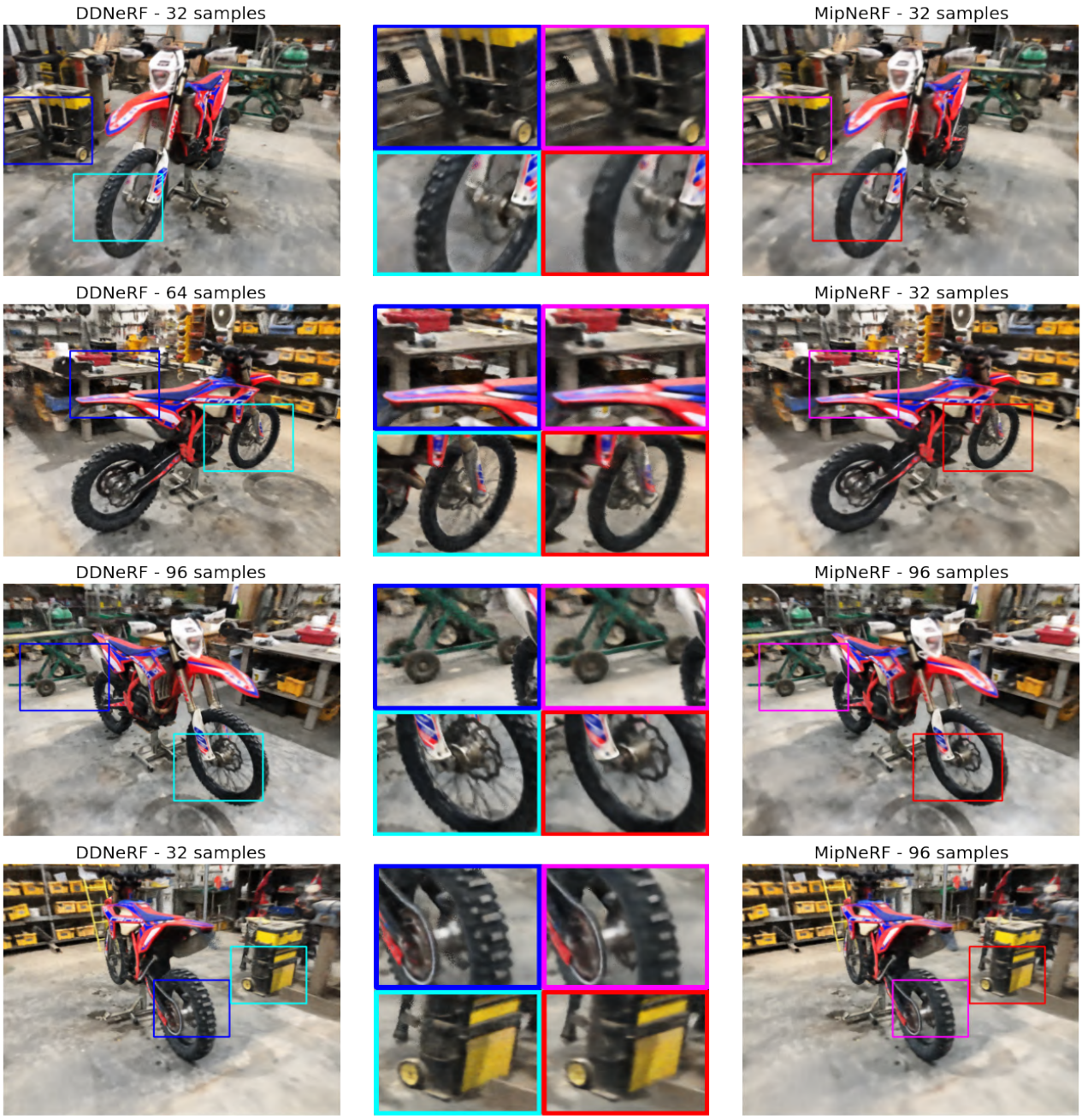


Figure 6. **Motorcycle results**: Left column – DDNeRF model. Right column – Mip-NeRF. The number of samples are noted above each image. Our model achieves better results for any number of samples. The colored crops in the middle column helps to visualize the differences between the results. Note, in the last row DDNeRF uses only one third of the number of samples while still outperforms Mip-NeRF.

We compare those GT values to the values predicted by our coarse model and Mip-NeRF coarse model. We calculate the relative errors from the coarse distributions $\mathbb{E}[h^c(t)]$ and $\mathbb{E}[f_{dd}(t)]$ to the GT points. We also calculate the coarse distribution s.t.d. Our model achieve lower error (i.e. the predicted $\mathbb{E}[f_{dd}(t)]$ is more accurate than Mip-NeRF predicted $\mathbb{E}[h^c(t)]$) and lower s.t.d (i.e. DDNeRF distribution focuses on smaller areas). The evaluation was performed using NDC space.

Figure 7. **3D reconstruction from point clouds**: The first row was reconstructed with Mip-NeRF. The second row was reconstructed with DDNeRF. Note, that DDNeRF achieves finer and less noisy shapes, especially around complex geometry regions. Notice, that the motorcycle lift was lost during the clustering in the Mip-NeRF model, which implies weaker structural connectivity.

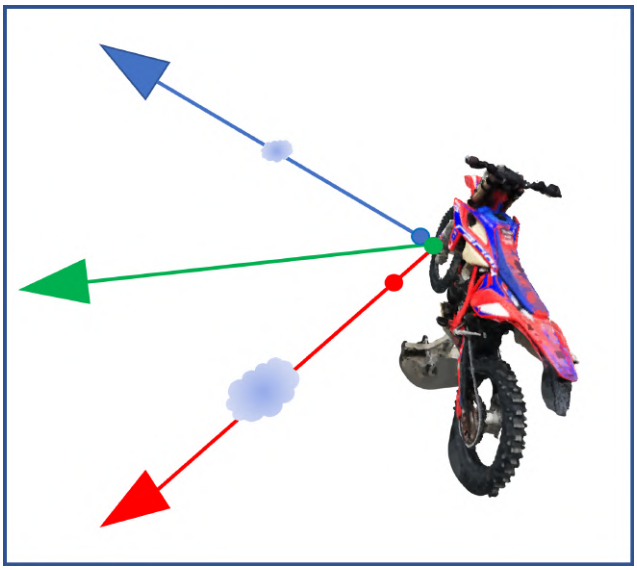

Figure 8. **Occlusion artifacts**: Same location in space may get non perfectly align points from 3 different views. The green camera has clean view to the point and the the green point is relatively accurate. The red and blue cameras dealing with big and small artifacts that cause the red and the blue point to diverge from the exact spatial location.

Table 2. **Ablation study:** Motorcycle scene, 32 samples, 300k iterations. We train with a constant variance value, without using the uncertainty factor, using a constant uncertainty factor, and finally our full model. The results show that predicting the variance is better than using a constant variance. The table also demonstrates the importance of the uncertainty factor to the learning process.

| Model | Version | PSNR↑ | SSIM↑ | LPIPS↓ |
|---|---|---|---|---|
| Mip-NeRF | 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 |

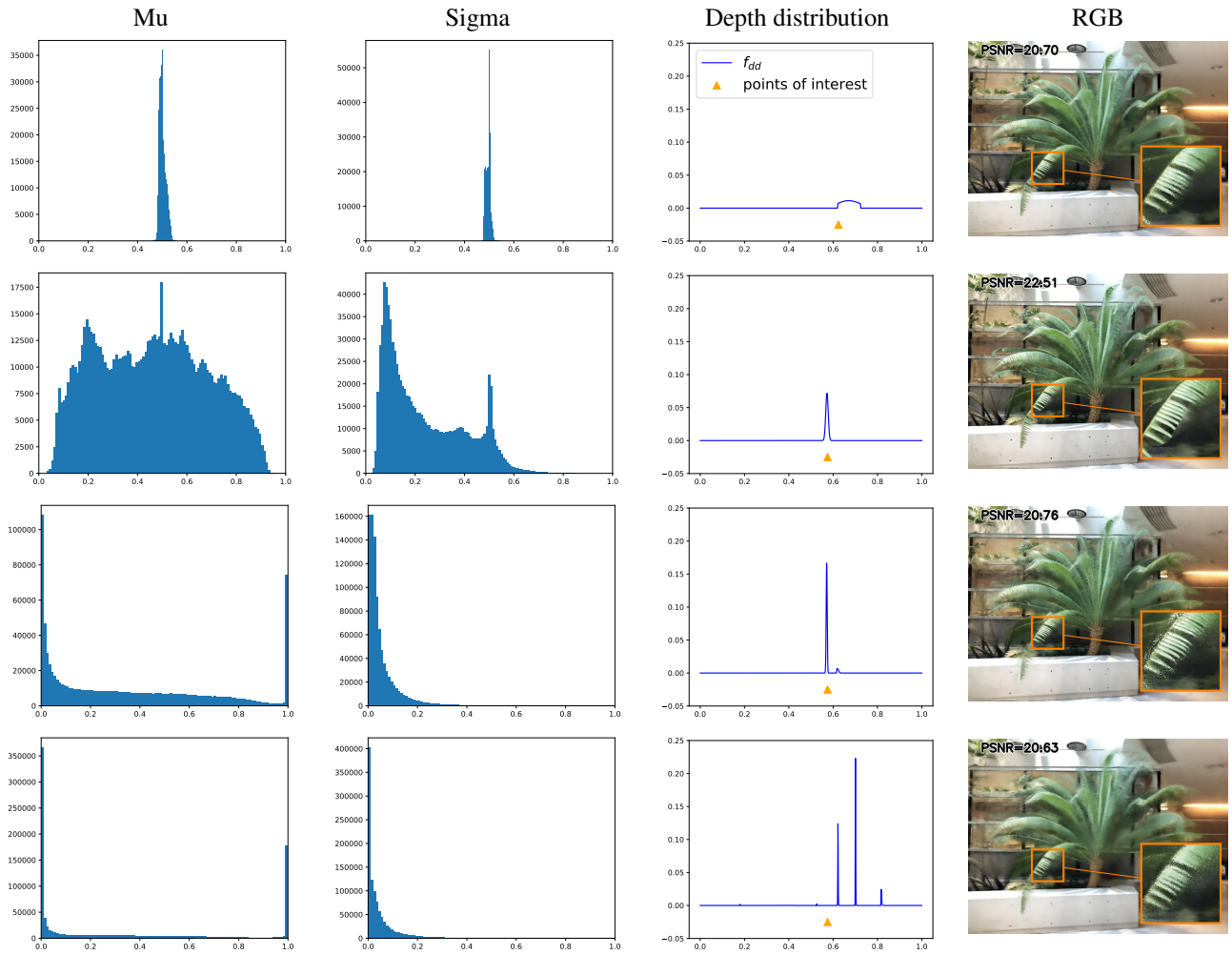


Figure 9. **Regularization effect**: First row – A model with too high regularization value. Second row – A model with uncertainty factor and good regularization value. Third row – A model with uncertainty factor and without regularization. Last row – A model without uncertainty factor and without regularization. The two left columns are the intervals $\mu$ and $\sigma$ histograms. The right column is the RGB output of the model with its PSNR value. The second column from the right is an example of the model density distribution along a single ray. Notice, how too high regularization factor cause $\mu$ and $\sigma$ values remain close to 0.5 and prevents efficient sampling procedure. On the other hand, notice that without regularization the $\sigma$ decrease toward zero, causing the Gaussian to over-shrink and reduce the output quality. Also, note how the $\mu$ is pushed toward 0 and 1 due to the sigmoid vanishing gradient areas.

Table 3. **Ablation study on model parameters number:** Motorcycle scene, 32 samples, 300k iterations. Coarse and Fine columns represents the number of neurons per hidden layer in each of the models. Sec/it is the time (in sec.) for one training iteration. The Render column measures the average rendering time for a 1008x756 image. Performance was measured on an RTX3090 GPU.

| Coarse | Fine | PSNR↑ | SSIM↑ | LPIPS↓ | Sec/it | Render | Mem |
|---|---|---|---|---|---|---|---|
| 256 | 256 | 20.63 | 0.572 | 0.446 | 0.066 | 7.10 | 1.84GB |
| 128 | 256 | 20.61 | 0.569 | 0.456 | 0.059 | 5.85 | 1.47GB |
| 64 | 256 | 20.94 | 0.571 | 0.474 | **0.056** | **5.19** | **1.28GB** |
| 128 | 338 | 20.97 | 0.596 | **0.420** | 0.068 | 7.29 | 1.73GB |
| 64 | 356 | **21.43** | **0.605** | 0.432 | 0.068 | 6.56 | 1.58GB |

Table 4. **Distribution accuracy:** Fern Scene with various number of samples. Error was calculated as the relative error between $\mathbb{E}[h^c(t)]$ and the GT point for Mip-NeRF and the relative error between $\mathbb{E}[f_{dd}(t)]$ and the GT point for DDNeRF. STD was calculated relative to the GT distance.

| Model | 4 Samples | | 8 Samples | | 16 Samples | |
|---|---|---|---|---|---|---|
| | Error | Std | Error | Std | Error | Std |
| Mip-NeRF | 8.34% | 10.36% | 7.91% | 6.58% | 1.83% | 3.68% |
| DDNeRF | **3.96%** | **2.96%** | **2.54%** | **2.69%** | **1.41%** | **2.39%** |

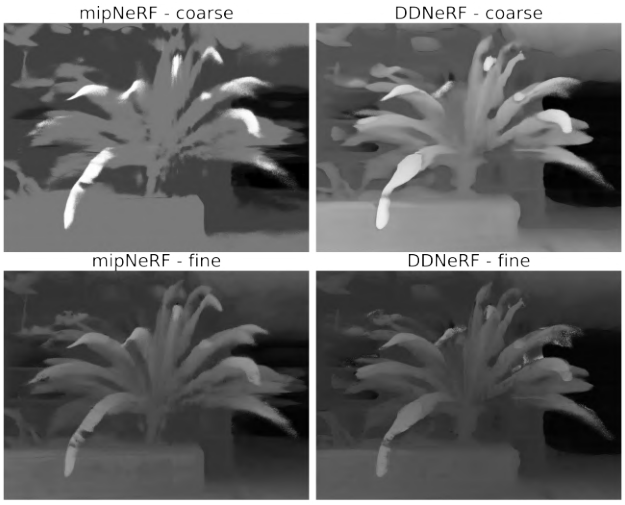


Figure 10. **Disparity comparisons:** The estimated disparity for the Fern scene measured for Mip-NeRF and DDNeRF. Both models were trained and evaluated using **eight samples** and without NDC warping. Notice how the depth estimation of our coarse model is closer to the estimation of the fine model, and is better than the estimation of the Mip-NeRF's coarse model.

## References


[1] 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.

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and*, pages 226–231, 1996.

[3] 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.

[4] 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.

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