

# WELL-NeRF: ENSURING WELL-POSED NEURAL RADIANCE FIELDS VIA VIEW FRUSTUM AND SHADOW ZONE BASED REGULARIZATION

**Anonymous authors**

Paper under double-blind review

## *Supplementary Material*

### 1 EXPERIMENTAL SETUP DETAILS (FIGURES & TABLES)

#### 1.1 FIG.1 (IN THE MAIN TEXT)

Fig.1(c): This is an image generated during the actual learning process with graphics added for understanding. It was generated through Nerfstudio's Viser viewer. In the figure below, the small image overlays the input view frustum onto the image generated through Viser viewer.

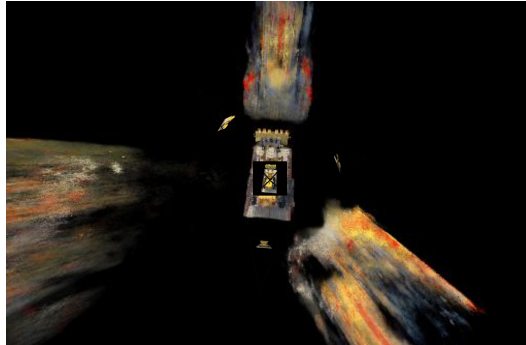


Figure 1: **Original image of Fig.1(c).**

#### 1.2 FIG.2 (IN THE MAIN TEXT)

No additional explanation

#### 1.3 FIG.3 (IN THE MAIN TEXT)

No additional explanation

#### 1.4 FIG.4 (IN THE MAIN TEXT)

For easy understanding, the top surface of the ground truth (GT) is represented in blue, but the color of the top surface and the inside of the object could appear red or black. The key point is that if the color of the surface and the inside of an unobserved object remains undecidable, it can cause instability during the learning process.

## 1.5 FIG.5 (IN THE MAIN TEXT)

- Data : Blender synthetic datasets Mildenhall et al. (2020)
- Input resolution :  $800 \times 800$  (original resolution)
- Train eval method : Nerfstudio train eval split method  
`pipeline.datamanager.dataparser.eval_mode = "fraction"`
- Number of input views :  
`pipeline.datamanager.dataparser.train_split_fraction =`  
`0.5(50), 0.2(20), 0.1(10), 0.07(7), 0.03(3)`
- Iteration : 10k

The original resolution was used and trained using only the data in the train folder. The 100 train data were split by a certain ratio and used for training and evaluation. For example, with a split ratio of 0.1, 10 images were used to training. The original image resolution was used for training. The model was trained for 10k iterations. The x-axis represents the split ratio : 0.5(50), 0.2(20), 0.1(10), 0.07(7), and 0.03(3). The values in parentheses indicate the number of images used for training.

## 1.6 FIG.6 (IN THE MAIN TEXT)

- Data : Blender synthetic datasets Mildenhall et al. (2020)
- Input resolution :  $800 \times 800$  (original resolution)
- Train eval method : Nerfstudio train eval split method  
`pipeline.datamanager.dataparser.eval_mode = "fraction"`
- Number of input views :  
`pipeline.datamanager.dataparser.train_split_fraction =`  
`0.07(7 inputs)`
- Iteration : 10k

Training was conducted with a split ratio of 0.07, using 7 out of 100 training images. The original image resolution was used for training, which was run for 10K iterations.

## 1.7 FIG.7 (IN THE MAIN TEXT)

The image shows the rendering results for the chair class from the experiments in Table 1.

## 1.8 TABLE 1 (IN THE MAIN TEXT)

- Data : Blender synthetic datasets Mildenhall et al. (2020)
- Input resolution :  $400 \times 400$  ( $2 \times$  downsampled) Yang et al. (2023)
- Train eval method : refer from FreeNeRF (preset train eval index) Yang et al. (2023)

- Number of input views : 8 input views (25 eval views)

```
train_indices = [26, 86, 2, 55, 75, 93, 16, 73]
                (start from\0")
```

```
eval_indices = [0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88,
                96, 104, 112, 120, 128, 136, 144, 152, 160,
                168, 176, 184, 192]
```

- Iteration :

```
FreeNeRF : 50k
ours      : 30k
```

For comparison on sparse input, we used the same experimental setup as FreeNeRF. The experiments on FreeNeRF were conducted using its official GitHub repository. The "train eval split" method followed the approach described in the original FreeNeRF paper, downsampled by  $2\times$ . FreeNeRF was trained for 50k iterations, while our model (and Nerfacto) was trained for 30k iterations.

## 1.9 TABLE 2 (IN THE MAIN TEXT)

- Data : Blender synthetic datasetsMildenhall et al. (2020)
- Input resolution :  $800 \times 800$  (original resolution)
- Train eval method : Nerfstudio train eval split method

```
pipeline.datamanager.dataparser.eval_mode = "fraction"
```

- Number of input views :

```
pipeline.datamanager.dataparser.train_split_fraction =
                0.9 (90 input images, 10 eval images)
```

- Iteration : 10k

We used 90 out of the 100 training data as input for training, with the remaining 10 used for evaluation.

## 2 ADDITIONAL DATA SET

### 2.1 DTU

- Data : DTU datasets Yu et al. (2021)
- Input resolution :  $300 \times 400$  ( $4\times$  downsampled form original DTU dataset)
- Train eval method : refer from FreeNeRF (preset train eval index)Yang et al. (2023), evaluation with object mask
- Number of input views : 9 input views (25 eval views)

```
train_indices = [25, 22, 28, 40, 44, 48, 0, 8, 13]
                (start from\0")
```

```
eval_indices = [1, 2, 9, 10, 11, 12, 14, 15,
                23, 24, 26, 27, 29, 30, 31, 32,
                33, 34, 35, 41, 42, 43, 45, 46, 47]
```

- Iteration :

```
FreeNeRF : 132k (refer from FreeNeRF)
ours      : 30k
```

Please note that the object mask was applied during evaluation, not during the learning process.

## 2.2 LLFF

- Data : LLFF datasets Mildenhall et al. (2020)
- Input resolution :  $378 \times 504$  ( $8\times$  downsampled)
- Train eval method : refer from FreeNeRF Yang et al. (2023)
- Number of input views : 6 input views
- Iteration :

```
FreeNeRF : 140k (refer from FreeNeRF)
ours      : 30k
```

## 3 METHOD IMPLEMENTATION DETAIL

### 3.1 MAJOR METHODS

#### 3.1.1 FRUSTUM SCORE CALCULATOR

The definition of the `isInside` function is shown below.

$$\text{isInside} = \forall i \in \{x, y, z\} : -\text{ClipSpacePoints}_{...,3} \leq \text{ClipSpacePoints}_{...,i} \leq \text{ClipSpacePoints}_{...,3}. \quad (1)$$

Then, `ClipSpacePoints` can be calculated as

$$\text{ClipSpacePoints} = \text{ViewSpacePoints} \cdot \text{ProjectionMatrix}^T, \quad (2)$$

where `ViewSpacePoints` and `Projection Matrix` are defined as follows.

$$\text{ViewSpacePoints} = \text{HomogeneousPoints} \cdot (\text{TransformMatrices}^{-1})^T, \quad (3)$$

and

$$\text{Projection Matrix} = \begin{bmatrix} \frac{1}{\tan(\frac{\text{fov}_x}{2}) \times \text{aspect\_ratio}} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\text{fov}_y}{2})} & 0 & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2 \times \text{far} \times \text{near}}{\text{far}-\text{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}. \quad (4)$$

### 3.1.2 CUSTUMGRAD

The implementation of our custom grad was inspired by the floater-no-more Philip & Deschaintre (2023) paper, which reduces the learning rate close to the camera. The approach is based on the difference in sampling rates in the unit space. In contrast, our implementation is based on our assumptions about regions where positional inference is not feasible.

When the overlap score of the view frustum reaches a certain level, it is already possible to make reasonable location inferences. Making a distinction in gradients for increases beyond this point can negatively impact learning. Therefore, scores above 9 are adjusted to 9. Furthermore, scores of 1 indicate regions where location inference is theoretically impossible; thus, we subtract 1 from these scores to reduce the gradient to 0. Then, we normalize this adjusted score for use in the backward pass.

$$f_{cg,Forward}(\mathbf{c}_\theta, \sigma_\theta, \mathbf{S}) = (\mathbf{c}_\theta^*, \sigma_\theta^*), \quad (5)$$

$$S_{adj} = \min(S, 9) - 1, \quad (6)$$

$$S_{norm} = \frac{S_{adjusted}}{S_{max}}, \quad (7)$$

$$f_{cg,Backward} \left( \frac{\partial L}{\partial \mathbf{c}}, \frac{\partial L}{\partial \sigma} \right) = \left( \frac{\partial L}{\partial \mathbf{c}} \odot \mathbf{S}_{norm}^2, \frac{\partial L}{\partial \sigma} \odot \mathbf{S}_{norm}^2 \right). \quad (8)$$

### 3.1.3 SHADOW ZONE

$$\text{Basic RGB blending formula: } C'_i = C_i w_i + C_{i-1} w_{i-1}. \quad (9)$$

The total sum of  $w_i$  is 1, which means the sum of  $w_i$  and  $w_{i-1}$  can be less than 1. This may lead to a continuous decrease in the value of  $C'_i$ , which can be more pronounced, especially when the values of  $w_i$  and  $w_{i-1}$  are small. To address this,  $C'_i$  should be adjusted by multiplying the reciprocal of  $w_i + w_{i-1}$ . This adjustment prevents a rapid decrease in  $C_i$  and consequently helps achieve a more natural blending. Additionally, for the first element of the sequence,  $C'_0$ , we use  $C_0$  directly.

$$C'_0 = C_0, \quad (10)$$

$$C'_i = (C_i w_i + C_{i-1} w_{i-1}) \times \frac{1}{w_i + w_{i-1}}. \quad (11)$$

## 3.2 MINOR METHODS

### 3.2.1 IGNORE BLACK AND WHITE

The RGB(0,0,0) color cannot occur in an unprocessed digital photo because all photos contain physical noise (electrical and thermal), but it is easily observed in synthetic images. RGB(1,1,1) occurs when the digital sensor is saturated by light. Since these regions have no information or are completely lost, making positional inference impossible. Therefore, we develop a method to exclude these regions from learning. We introduce an additional regularization method to reduce the density of black or saturated parts in the GT image. This is implemented similarly to our frustum mask regularization method. This newly calculated loss term is added to the total loss, with the same dynamic lambda applied.

$$\sigma_{\text{masked}} = \sigma \odot \mathbf{M}, \quad (12)$$

where

$$M = \begin{cases} 0, & \text{if } \text{lower\_threshold} < C_{gt} < \text{upper\_threshold} \\ 1, & \text{otherwise,} \end{cases} \quad (13)$$

$$\mathcal{L}_{\text{ibw}} = \text{MSE}(\sigma_{\text{masked}}, \mathbf{0}). \quad (14)$$

It is also used by FreeNeRF in combination with its occlusion regularization method. However, while FreeNeRF introduces it as a solution to the wall appearance phenomenon, we apply it based on our understanding of regions where inference is not feasible.

### 3.2.2 IGNORE OUT OF INPUT VIEW FRUSTUM

We observe that rendering beyond the training view frustum causes numerous artifacts, a phenomenon also noted in Nerfbusters Warburg et al. (2023). Regions outside of the input view frustum cannot contribute to learning; they merely extend the model’s fitting results in the training region. However, as the training region is fitted more precisely, the more complex noise outside this region is generated, which is completely irrelevant to our ROI. Therefore, we exclude the input view frustum sampling points from rendering, as they do not contribute to learning and only degrade rendering quality.

$$\sigma_i = \begin{cases} 0, & \text{if } \text{scores}_i = 0 \\ \sigma_i, & \text{otherwise,} \end{cases} \quad (15)$$

$$RGB_i = \begin{cases} \mathbf{0}, & \text{if } \text{scores}_i = 0 \\ C_i, & \text{otherwise.} \end{cases} \quad (16)$$

Please note that this method applies only to the rendering process and does not affect the training process.

The difference between Nerfbusters and our method is that Nerfbusters directly samples areas outside the input view frustum by generating random ray and calculates visibility loss, which directly influences learning. In contrast, our method only applies it to the rendering process and does not affect model learning.

## 4 DETAIL EXPLANATION OF NERFACTO

We explore the intricate details of the Nerfacto model, a sophisticated extension of the NeRF architecture. This model integrates advanced techniques from both Mip-NeRF 360 and Instant-NGP, incorporating features such as scene contraction, multi-resolution hash encoding, and spherical harmonics encoding. These enhancements significantly improve visual accuracy and computational efficiency.

### 4.1 SCENE CONTRACTION

As introduced by Mip-NeRF360, scene contraction deals with unbounded scenes in the real world by compressing them into a bounding box. This approach uses the  $L^\infty$  norm to contract a cube rather than a sphere, enhancing compatibility with voxel-based hash encoding. This scene contraction process  $f_{sc}(x)$  in equation 17 maps infinite samples  $x$  into a range from -2 to 2 and is employed in conjunction with the hash encoding introduced in Instant-NGP.

$$f_{sc}(x) = \begin{cases} x & \text{if } \|x\|^\infty \leq 1, \\ (2 - \frac{1}{\|x\|^\infty}) \left( \frac{x}{\|x\|^\infty} \right) & \text{if } \|x\|^\infty > 1. \end{cases} \quad (17)$$

### 4.2 PROPOSAL MLP

Proposal MLP facilitates the efficient training and rendering of NeRF models. Proposal MLP  $f_\delta$  in equation 19 takes the input ray origin  $\mathbf{o}$  and direction  $\mathbf{d}$  from generated ray  $\mathbf{x}(t)$  in equation 18 and identifies ROI in a scene by predicting volume density. It is trained using a weight histogram generated by the NeRF MLP. This structure provides a new sampling interval for the NeRF MLP that renders the final image, significantly increasing the capacity of the overall model while only slightly extending the training time.

$$\mathbf{x}(t) = \mathbf{o} + t\mathbf{d}, \quad (18)$$

where  $\mathbf{o}$  and  $\mathbf{d}$  denote the ray origin and direction, respectively.

$$\mathbf{x}_\delta = f_\delta(\mathbf{o}, \mathbf{d}). \quad (19)$$

### 4.3 MULTI-RESOLUTION HASH ENCODING

Multi-resolution Hash Encoding in equation 20 encodes the position of a sampling point into a lattice with different resolutions. This technique is designed to preserve fine details, optimize memory usage, and improve computational efficiency. The method divides the input space into grids of different resolutions and indexes feature vectors using a hash function for each grid level. This multi-resolution approach is particularly effective for modeling complex 3D scenes that contain a mixture of structures with varying sizes.

$$\hat{\mathbf{x}} = \text{HashEncoding}(\mathbf{x}_\delta) = \bigoplus_{\ell=1}^L \text{interp} \left( h \left( \lfloor \mathbf{x}_\delta \cdot N_\ell \rfloor, \lceil \mathbf{x}_\delta \cdot N_\ell \rceil \right) \right), \quad (20)$$

where  $h$  denotes a spatial hash function,  $N_\ell$  indicates the  $\ell$ -th resolution, and  $N$  is the total number of resolutions.

#### 4.4 SPHERICAL HARMONICS ENCODING

In NeRF training, the directions of the sampled points are encoded using Spherical Harmonics. Spherical harmonics Green (2003) can efficiently express the reflection and radiation characteristics according to frequency. These encoded directions are used as input to the NeRF MLP  $f_\theta$  in equation 22.

$$\hat{\mathbf{d}} = \text{SHEncoding}(\mathbf{d}) = (Y_\ell^m(\mathbf{d}))_{\ell: 0 \leq \ell \leq \ell_{\max}, m: -\ell \leq m \leq \ell}, \quad (21)$$

where  $Y_\ell^m$  denotes the spherical harmonic function.

$$\mathbf{c}_i, \sigma_i = f_\theta(\hat{\mathbf{x}}, \hat{\mathbf{d}}), \quad (22)$$

where  $\mathbf{c}_i$  and  $\sigma_i$  denote the color and density values for the  $i$ -th point along the ray, respectively.

#### 4.5 VOLUME RENDERING

NeRF’s volume rendering process first uses Proposal MLP in equation 19 to sample points above the ray. Each of these sampled points contains position  $\mathbf{x}$  and direction  $\mathbf{d}$  in equation 18. These points are then encoded and fed into the NeRF MLP in equation 22. The NeRF MLP outputs color  $\mathbf{c}$  and density  $\sigma$  values for each point along the ray. These outputs are used to calculate the transmittance in equation 23 and the weight of each point in equation 24. The transmittance and weight are calculated based on the density of the points, which is then used to obtain a weighted sum of the colors for the ray. This process determines the color of the final pixel, enabling the reconstruction of a complex 3D scene into a high-resolution 2D image.

$$T_i = \exp \left( - \sum_{j=1}^{i-1} \sigma_j \Delta t_j \right), \quad (23)$$

where  $T_i$  denotes the transmittance values for the  $i$ -th point along the ray.

$$C(\mathbf{r}) = \sum_{i=1}^N w_i \mathbf{c}_i \text{ and } w_i = T_i (1 - \exp(-\sigma_i \Delta t_i)), \quad (24)$$

where  $C(\mathbf{r})$  calculates a weighted sum of the colors for the ray.

## 5 LIMITED EXPERIMENTS ON DISTANT BACKGROUNDS OR COMPLEX SCENES

We conducted additional validation on the MipNeRF-360 dataset and our custom indoor dataset. When using only 5% of the entire dataset, we observed that the training error was eliminated and a significant improvement in performance was achieved (Fig.2) Our initial motivation was to improve the performance of 3D reconstruction for complex real-world indoor scenes. Although our experiments with our own data did not achieve the highest quality, our approach substantially





Frederik Warburg, Ethan Weber, Matthew Tancik, Aleksander Holynski, and Angjoo Kanazawa. Nerfbusters: Removing ghostly artifacts from casually captured nerfs. In *ICCV*, 2023.

Jiawei Yang, Marco Pavone, and Yue Wang. Freenerf: Improving few-shot neural rendering with free frequency regularization. In *CVPR*, 2023.

Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021.