
Analyzing the Internals of Neural Radiance Fields

Lukas Radl
radl@student.tugraz.at

Andreas Kurz
andreas.kurz@icg.tugraz.at

Markus Steinberger
steinberger@icg.tugraz.at

Graz University of Technology
<https://github.com/r4dl/nerf-internals>

Abstract

Modern Neural Radiance Fields (NeRFs) learn a mapping from position to volumetric density via proposal network samplers. In contrast to the coarse-to-fine sampling approach with two NeRFs, this offers significant potential for speedups using lower network capacity as the task of mapping spatial coordinates to volumetric density involves no view-dependent effects and is thus much easier to learn. Given that most of the network capacity is utilized to estimate radiance, NeRFs could store valuable density information in their parameters or their deep features. To this end, we take one step back and analyze large, trained ReLU-MLPs used in coarse-to-fine sampling. We find that trained NeRFs, Mip-NeRFs and proposal network samplers map samples with high density to local minima along a ray in activation feature space. We show how these large MLPs can be accelerated by transforming the intermediate activations to a weight estimate, without any modifications to the parameters post-optimization. With our approach, we can reduce the computational requirements of trained NeRFs by up to 50% with only a slight hit in rendering quality and no changes to the training protocol or architecture. We evaluate our approach on a variety of architectures and datasets, showing that our proposition holds in various settings.

1 Introduction

Neural Fields [34] parameterize implicit functions over continuous domains using neural networks. To represent high-frequencies, positional encoding [15, 28] or periodic non-linearities [23] are used to overcome the spectral bias [20] present in neural networks. Such implicit representations are more memory-efficient than their explicit counterparts and can be optimized with first-order optimization methods such as gradient descent. However, as the scene is encoded in the weights of a neural network, every change to a parameter has a very non-local effect, which results in poor interpretability.

For novel view synthesis problems, the 5D plenoptic function mapping positions and directions to radiance and density can be stored in such a neural field. NeRFs [15] demonstrate impressive performance for real-world and synthetic data, leveraging an implicit neural representation (INR). However, their approach is slow and inefficient due to the prohibitive amount of samples needed to evaluate the scene densely. To mitigate this, a coarse-to-fine sampling strategy is used, estimating the density distribution along a ray by sampling uniformly with a coarse NeRF, and then producing a new set of samples for a subsequent fine NeRF with inverse transform sampling. Using this approach, performance can be improved significantly, but the amount of samples needed still prohibits real-time

rendering of NeRFs. Further, the coarse NeRF is supervised with an MSE loss only, which is necessary as no ground-truth volumetric density is available.

Reducing the required amount of samples for the shading network is a common approach to lower the computational requirements. For this task, sampling networks [1, 3, 11, 17] prove fruitful, which predict volumetric density along a ray to produce a set of samples for a subsequent shading network. The network capacity required for a sampling network is lower compared to a standard NeRF, as the task of predicting volumetric density involves no view-dependent effects and is thus independent of the viewing direction, and easier to learn. For supervising these sampling networks ground-truth depth data [17], a pre-trained NeRF [19], multi-view stereo [12] or distillation from a concurrently trained shading network [1, 3] may be used.

A clear disadvantage of the coarse-to-fine NeRF pipeline is the use of the coarse model for density prediction only, even though it is optimized to reconstruct the ground-truth image. Sampling networks, on the other hand, map sample positions to volumetric density using a network with reduced capacity and can be supervised by the shading network. To address this limitation, we analyze the intermediate activations of NeRFs and show how to extract an approximation of volumetric density from this representation. We demonstrate that our approach applies to both NeRFs as well as proposal network samplers. To our knowledge, we are the first to use intermediate activations to speed up inference for coordinate-based MLPs.

In total, our contributions are as follows:

- We present a method for visualizing and analyzing the intermediate activations of coordinate-based ReLU-MLPs.
- We present an approach for extracting a density estimate for fine re-sampling from activations in early layers of coarse NeRFs, significantly reducing the inference time with little decrease in quality. We find this approach also applies to Mip-NeRFs and proposal sampling networks.
- We validate our method for several architectures with real-world and synthetic datasets.

2 Related Work

Our analysis is aimed at improving the interpretability of NeRFs, enabling faster inference after optimization. Therefore, we will consider different architectures for real-time rendering with radiance fields. We additionally discuss other methods used to improve the sampling efficiency of NeRFs.

NeRFs for Real-time Rendering Instead of the implicit representation of radiance fields used in NeRFs, explicit representations [5, 25] with grid-like structures or hybrid representation [4, 9, 16, 33, 35] are often used to speed up performance. Most prominently of these related works, Instant-NGP [16] replaces the fixed positional encoding used in [15] with a learnable hash input encoding in a pyramid of grids, allowing for the use of smaller MLPs and rapid convergence. Orthogonally, Garbin *et al.* [6] cache the radiance map and query it using ray directions. Several works [7, 21, 22] suggest a divide-and-conquer approach, utilizing many tiny MLPs responsible for a small region instead of a large MLP for the whole scene. A concurrent trend is to extract approximate geometry, compute features with ray-triangle intersection and use a shallow MLP on top [10, 30].

Efficient Sampling Several works investigate a learnable proposal sampler, which can predict volumetric density [3] or sample locations [1] directly. DONeRF [17] learns a density volume as a multi-class classification task to guide sample placement with depth supervision. AdaNeRF [11] builds upon DONeRF and mitigates the need for pre-training and supervision with a complex, 4-phase training recipe. EfficientNeRF [8] proposes valid sampling, caching densities of sample positions and exclusively evaluating those with positive density. This approach is, similar to our work, motivated by the density distribution of coarse NeRFs. TermiNeRF [19] distills a trained NeRF into a sampling and shading network. For their sampling network, they perform distribution matching, reparameterizing the NeRF intervals into smaller intervals, allowing for fewer samples. Lin *et al.* [12] learn a proxy geometry with a cost volume for depth guidance, uniformly sampling a smaller depth range with a lower sample count. A better sampling strategy is also attained by unifying implicit surface models and radiance fields [18].

3 Preliminaries

In the following section, we briefly summarize the most important architectures and concepts which build the foundation for our work.

3.1 NeRF

NeRFs [15] learn a function

$$\Theta_{\text{NeRF}} : \mathbb{R}^5 \rightarrow \mathbb{R}^4, (\mathbf{p}, \mathbf{d}) \mapsto (\mathbf{c}, \sigma), \quad (1)$$

where $\mathbf{p} = (x, y, z)$ and $\mathbf{d} = (\theta, \phi)$ are the sample position and viewing direction, $\mathbf{c} = (\textcolor{red}{r}, \textcolor{green}{g}, \textcolor{blue}{b}) \in [0, 1]^3$ denotes the predicted output color and $\sigma \in \mathbb{R}$ denotes the predicted volumetric density. NeRF uses a ReLU-MLP with 8 layers and 256 hidden units to predict a feature vector $\mathbf{f} \in \mathbb{R}^{256}$ and σ . This feature vector is then concatenated with \mathbf{d} and passed through a small ReLU-MLP to produce \mathbf{c} , which allows view-dependent phenomena to be modelled. For each pixel, we cast a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ from the origin \mathbf{o} in the direction of \mathbf{d} . We evaluate Θ_{NeRF} at different sample positions $t \in [t_n, t_f]$ between the near bound t_n and the far bound t_f . The color $\hat{\mathcal{C}}$ of a ray \mathbf{r} can be estimated using N_s samples with

$$\hat{\mathcal{C}}(\mathbf{r}) = \sum_{i=1}^{N_s} T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i, \quad (2)$$

where δ_i and T_i are given as

$$\delta_i = t_{i+1} - t_i, \quad T_i = \exp \left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right). \quad (3)$$

To increase rendering efficiency, two NeRFs are concurrently optimized. This avoids placing a large number of samples in non-occupied space. The coarse NeRF Θ_{coarse} places N_c samples uniformly for each ray \mathbf{r} , which produces a set of weights $\mathbf{w} \in \mathbb{R}^{N_c}$ with

$$\mathbf{w}_i = T_i (1 - \exp(-\sigma_i \delta_i)), \quad (4)$$

used to express the estimated coarse radiance as a weighted sum

$$\hat{\mathcal{C}}_c(\mathbf{r}) = \sum_{i=1}^{N_c} \mathbf{w}_i \mathbf{c}_i. \quad (5)$$

We normalize \mathbf{w} such that $\mathbf{1}^T \mathbf{w} = 1$, producing a piecewise-constant PDF along \mathbf{r} . This distribution is used to produce a new set of N_f samples with inverse transform sampling. The final rendered color is produced by Θ_{fine} using all $N_c + N_f$ samples.

The authors apply positional encoding $\gamma(\cdot)$ to \mathbf{p} and \mathbf{d} , allowing the NeRF to model high-frequency details effectively [26]. The network is supervised with the MSE loss for both Θ_{coarse} and Θ_{fine} , i.e. $\sum_{\mathbf{r}} \|\hat{\mathcal{C}}_c(\mathbf{r}) - \mathcal{C}(\mathbf{r})\|_2^2 + \|\hat{\mathcal{C}}_f(\mathbf{r}) - \mathcal{C}(\mathbf{r})\|_2^2$, with $\mathcal{C}(\mathbf{r})$ denoting the ground truth color for \mathbf{r} and $\hat{\mathcal{C}}_f(\mathbf{r})$ denoting the estimated fine radiance of \mathbf{r} .

3.2 Mip-NeRF

Mip-NeRF [2] builds on NeRF and tackles the problems of blurring and aliasing when scene content is observed at various resolutions. The standard computer graphics solution of super-sampling is prohibitively expensive with NeRF's volume rendering. The authors introduce a novel integrated positional encoding (IPE), encoding a region of space rather than a point in space. Instead of casting a single ray per pixel, they cast a cone from \mathbf{o} through the pixel and divide it into conical frustums. The conical frustums are approximated with a multivariate Gaussian, and their expected IPE feature is computed, which serves as the encoded input for Θ_{NeRF} .

As the network can now learn a multi-scale scene representation, Θ_{NeRF} performs both coarse and fine sampling, reducing the model size by 50%. The network architecture is similar to NeRF described above, with the main differences being the single NeRF MLP used and a hyperparameter $\lambda = 0.1$, balancing coarse and fine loss.

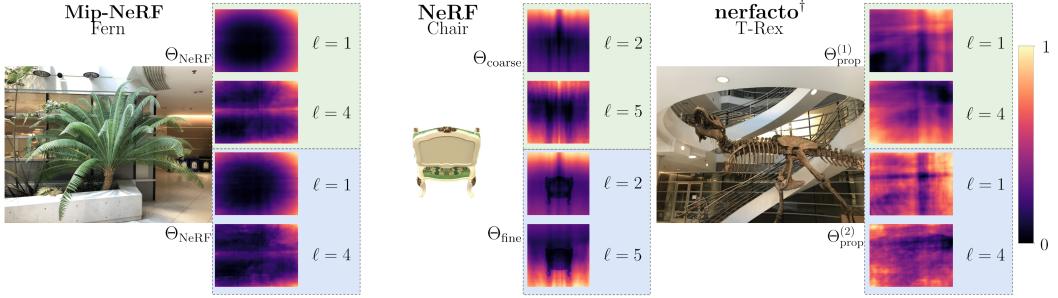


Figure 1: Rendered ground truth images from the test set and their corresponding normalized coarse and fine activations v_ℓ using the *magma* colormap. We visualize different layers ℓ for each view. For the nerfacto^\dagger model, we visualize activations for both rounds of the proposal network sampler.

3.3 Mip-NeRF 360

NeRF and Mip-NeRF work well on forward-facing, bounded scenes. As an extension to unbounded, 360° captures of a scene, Mip-NeRF 360 [3] proposes a different scene parameterization, warping distant objects towards 0 and sampling linearly in disparity. For better sampling efficiency, which is crucial in unbounded scenes, the authors concurrently optimize two MLPs, Θ_{prop} and Θ_{NeRF} .

Their proposal network sampler learns a function

$$\Theta_{\text{prop}} : \mathbb{R}^3 \rightarrow \mathbb{R}, \mathbf{p} \mapsto \sigma, \quad (6)$$

and is optimized to bound the weights w produced by Θ_{NeRF} . The authors use two subsequent rounds of sampling to produce two weights $\{\hat{w}_1, \hat{w}_2\}$, which is efficient due to the small size of Θ_{prop} with 4 layers and 256 hidden units. In addition, Mip-NeRF 360 introduces a novel regularizer to prevent common NeRF artefacts such as floaters by forcing the density along a ray to be concentrated in a narrow interval.

4 Activation Analysis

In this section, we present our method for visualizing and analyzing the intermediate activations for coordinate-based MLPs. Additionally, we propose methods for speeding up inference using our obtained representations.

4.1 Visualizing Activations

For Convolutional Neural Networks (CNNs), various approaches for visualizing intermediate features exist [24, 37]. These methods work by back-projection from feature space to pixel space, exploiting the spatial structure baked into the building blocks of CNNs. Traditional MLPs, on the other hand, are merely universal function approximators and operate mostly as a black box.

We analyze the hidden layer activations of NeRF MLPs as follows: For each pixel (x, y) , a ray \mathbf{r} is generated, positional encoding $\gamma(\cdot)$ is applied and each sample along the ray is passed through Θ_{NeRF} . In the following, we describe the intermediate activations in terms of a single ray/pixel. After each linear layer, the activation $\mathbf{A}^{(\ell)}$ is of size $\mathbb{R}^{N_s \times N_h}$, where N_s and N_h denote the number of samples and the number of hidden units, respectively. To obtain a feature representation for the activation in layer ℓ , we compute the mean over the dimension N_h :

$$\mathbf{f}_\ell = \frac{1}{N_h} \sum_{i=1}^{N_h} \mathbf{A}_i^{(\ell)}. \quad (7)$$

Further, we can produce a pixel value for each ray \mathbf{r} for visualization purposes, using

$$v_\ell = \frac{1}{N_s N_h} \sum_{i=1}^{N_s} \sum_{j=1}^{N_h} \mathbf{A}_{i,j}^{(\ell)}. \quad (8)$$

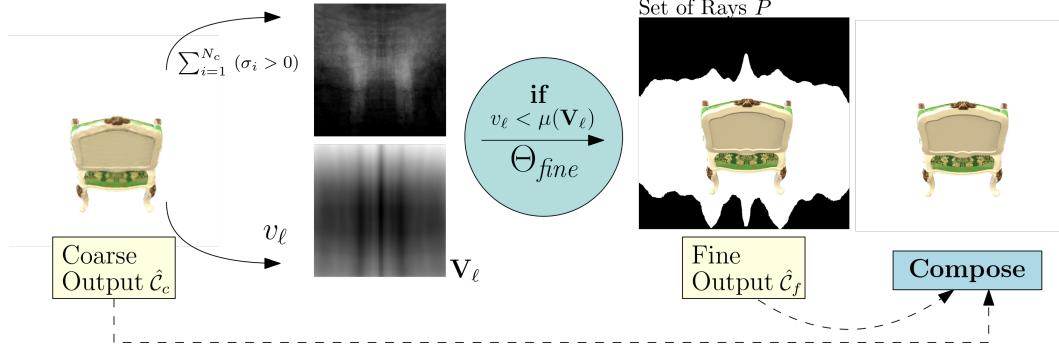


Figure 2: Using intermediate activations allows for simple performance improvements. We can reduce the inference time of NeRF in synthetic scenes if we perform the fine pass only if the condition $v_\ell < \mu(\mathbf{V}_\ell)$ is fulfilled.

Table 1: Quantitative results for our activation informed coarse-to-fine experiment. Only performing the fine pass for rays \mathbf{r} where the summed activation is smaller than $\tau = \mu(\mathbf{V}_\ell)$ yields a significant performance gain with a slight drop in rendering quality. We evaluated the *lego* scene of the Blender [15] dataset using a NeRF. Results were obtained on an NVIDIA RTX 2070 Super with a batch size of 1024 rays.

ℓ	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	t [s]	Speedup	% fine Rays
1	29.66 ± 1.21	0.93 ± 0.01	0.04 ± 0.01	34.69 ± 0.56	40.01%	59.27%
2	30.45 ± 1.07	0.94 ± 0.01	0.03 ± 0.01	34.02 ± 1.18	42.78%	57.59%
3	31.21 ± 1.26	0.95 ± 0.01	0.02 ± 0.01	33.11 ± 1.34	46.68%	55.03%
4	30.71 ± 1.68	0.95 ± 0.01	0.03 ± 0.01	34.76 ± 1.91	42.59%	59.44%
NeRF	32.20 ± 1.19	0.96 ± 0.00	0.02 ± 0.00	48.98 ± 0.50	0%	100%

We visualize the activations of different layers ℓ for some example views using Eqn. (8) in Fig. 1. For some visualizations, we can already infer some scene information with our proposed method. We note that when tracking activations $\mathbf{A}^{(\ell)}$, we have the option to apply or not apply the activation function. Unless stated otherwise, we work with activations after the ReLU has been applied, thus $\mathbf{A}^{(\ell)} \in \mathbb{R}_{+}^{N_s \times N_h}$. Early experiments showed that applying the activation function resulted in more consistent histograms for \mathbf{f}_ℓ .

4.2 Reducing Unnecessary Network Evaluations using Activations

As a first step towards speeding up the coarse-to-fine NeRF rendering pipeline, we analyze the densities σ predicted by Θ_{coarse} . Take, for example, any of the synthetic scenes from the Blender dataset [15]. Along rays \mathbf{r} in empty space, we do not expect any sample with significant volumetric density. We can sum the number of densities along each ray \mathbf{r} larger than a small threshold τ to obtain a mapping $g : \mathbf{r} \mapsto \{0, \dots, N_c\}$. If for any of the rays the result is 0, the corresponding pixel likely belongs to the transparent background. We note that this visualization behaves inversely to the activation v_ℓ , as seen in Fig. 2.

For each ray \mathbf{r} , we record the predicted coarse radiance \hat{c}_c and the activations v_ℓ from Θ_{coarse} . We choose a threshold τ , which indicates whether we observe scene content along \mathbf{r} . We choose $\tau = \mu(\mathbf{V}_\ell)$, so the mean activation of the whole activation image \mathbf{V}_ℓ . We obtain a set of rays $P = \{\mathbf{r} : v_\ell < \tau\}$, which we treat as a mask. For each $\mathbf{r} \in P$, we evaluate Θ_{fine} . For all $\mathbf{r} \notin P$, we take the coarse radiance \hat{c}_c as our final output. Using this approach, we can significantly reduce the number of evaluations for Θ_{fine} , which uses $N_c + N_f = 192$ samples, resulting in a significant speedup. As we cover most of the content with our derived mask P , the image quality loss is minimal. We report the results for the *lego* scene in Tab. 1 and visualize our approach in Fig. 2.

We note that we could use σ directly as we perform the full coarse pass for each pixel to get the coarse radiance \hat{c}_c . Also, this approach is highly susceptible to the synthetic scenes of the Blender

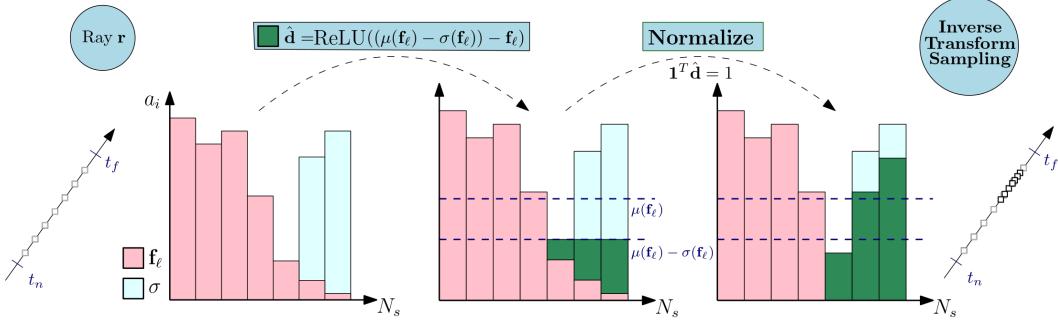


Figure 3: Visualization of our density estimation approach on a small toy example with $N_s = 7$. We find that local minima in activation feature space map to samples with high density. Transforming the activation features with any of our proposed functions achieves a good density estimate \hat{d} . For this example, we use Eqn. (9).

dataset. Thus, the use of activations is not necessary, but this serves as a first example on the utility of our approach.

4.3 Approximating Density using Activations

We go beyond this basic approach by analyzing the activation features f_ℓ along a ray r . We accomplish this by examining histograms of the predicted density σ and the activation features f_ℓ . We notice a trend throughout multiple scenes: Activation feature space minima indicate samples with significant density along the ray r . If we can find a function to transform f_ℓ to a reasonable density estimate \hat{d} , we can skip virtually the entire forward pass for Θ_{coarse} . Ideally, this function should transform f_ℓ to a vector of mostly zeros, except for locations where σ is significant. To this end, we propose 3 different functions to map the intermediate activations directly to estimated densities \hat{d} :

$$\hat{d} = f_1(f_\ell) = \text{ReLU}((\mu(f_\ell) - \sigma(f_\ell)) - f_\ell), \quad (9)$$

$$\hat{d} = f_2(f_\ell) = \text{ReLU}\left(\left(\mu(f_\ell) - \frac{\sigma(f_\ell)}{2}\right) - f_\ell\right), \quad (10)$$

$$\hat{d} = f_3(f_\ell) = \text{ReLU}\left(\left(\mu(f_\ell) - \frac{\sigma(f_\ell)}{2}\right)^2 - f_\ell\right), \quad (11)$$

where f_ℓ denotes the intermediate activation in layer ℓ and $\mu(\cdot)$, $\sigma(\cdot)$ denote the mean/std. deviation along the last dimension (i.e. along the N_s samples of the ray r). We visualize our approach in a toy example in Fig. 3.

Using Eqn. (9) results in fewer false positives along the ray, but might cause catastrophic failures if the distribution along the ray yields a high standard deviation, which produces no predicted density along the ray. On the other hand, Eqns. (10) and (11) extract densities more conservatively, but might lead to insufficient detail, as the interval between fine samples grows larger.

Given a well-behaved distribution, we can estimate densities sufficiently close to σ such that we get good quality reconstruction when converting \hat{d} to a weight estimate \hat{w} . Further, we can significantly reduce the required inference time by using only a partial pass through Θ_{coarse} . For further motivation behind these functions, we provide more details in the supplementary material.

5 Experiments

In the following, we apply our presented method to NeRF [15], Mip-NeRF [2] and proposal network samplers, utilizing a nerfacto [27] model with a Mip-NeRF 360 [3] proposal sampler. We use the Blender [15] dataset and the LLFF [14] dataset for our experiments. For the Blender dataset, we render the images at a resolution of 800×800 and use all 200 images from the test set. Concerning the LLFF dataset, we render at a resolution of 1008×756 . We use 90% of images for training and the remaining 10% for testing. We compensate for the different number of test images per scene by

Table 2: Quantitative evaluation of our approach for NeRF, Mip-NeRF and nerfacto[†]. For each dataset, we report PSNR with mean \pm std. deviation, SSIM [31] and LPIPS [38]. We highlight **best**, **second-best** and **third-best** for each image metric. Additionally, we point out **failures** of our method.

NeRF												
ℓ	f	Blender, 800 \times 800					LLFF, 1008 \times 756					
		PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	t [s]	Speedup	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	t [s]	Speedup	
1	f_1	27.48 \pm 1.19	0.92	0.09	25.41	23.9%	25.11 \pm 0.62	0.75	0.23	27.94	28.1%	
	f_2	28.69 \pm 1.10	0.92	0.08	25.41	23.9%	25.40 \pm 0.66	0.76	0.22	27.93	28.1%	
	f_3	28.63 \pm 1.08	0.92	0.08	25.42	23.8%	25.34 \pm 0.65	0.75	0.23	28.02	27.7%	
2	f_1	27.99 \pm 1.73	0.92	0.08	26.21	20.1%	24.72 \pm 0.57	0.73	0.26	28.85	24.1%	
	f_2	29.05 \pm 1.14	0.93	0.07	26.22	20.0%	25.12 \pm 0.62	0.74	0.24	28.89	23.9%	
	f_3	29.44 \pm 1.16	0.93	0.07	26.22	20.0%	25.01 \pm 0.61	0.74	0.25	28.87	24.0%	
3	f_1	24.17 \pm 2.45	0.88	0.12	26.67	18.0%	24.21 \pm 0.49	0.71	0.28	29.76	20.3%	
	f_2	26.23 \pm 1.45	0.90	0.11	26.70	17.9%	24.99 \pm 0.59	0.74	0.25	29.86	19.9%	
	f_3	25.07 \pm 1.75	0.89	0.11	26.70	17.9%	24.87 \pm 0.58	0.73	0.26	29.80	20.1%	
NeRF		29.87 \pm 1.25	0.94	0.06	31.48	0.0%	25.89 \pm 0.73	0.78	0.18	35.80	0.0%	
Mip-NeRF												
1	f_1	27.54 \pm 1.45	0.92	0.09	21.97	58.4%	25.07 \pm 0.66	0.74	0.24	29.12	50.4%	
	f_2	28.75 \pm 1.15	0.92	0.08	21.99	58.3%	25.45 \pm 0.70	0.76	0.22	29.08	50.7%	
	f_3	28.72 \pm 1.15	0.93	0.08	21.98	58.3%	25.35 \pm 0.69	0.75	0.22	28.93	51.4%	
2	f_1	26.67 \pm 1.82	0.91	0.09	23.38	48.8%	25.02 \pm 0.68	0.74	0.25	31.09	40.9%	
	f_2	29.04 \pm 1.17	0.93	0.07	23.40	48.7%	25.44 \pm 0.70	0.75	0.22	31.28	40.1%	
	f_3	29.35 \pm 1.19	0.93	0.07	23.40	48.7%	25.35 \pm 0.69	0.75	0.23	31.51	39.1%	
3	f_1	23.47 \pm 4.45	0.90	0.10	25.05	38.9%	24.21 \pm 0.49	0.71	0.28	29.76	20.3%	
	f_2	27.47 \pm 1.54	0.92	0.09	25.06	38.8%	25.49 \pm 0.70	0.76	0.22	32.14	36.3%	
	f_3	26.45 \pm 1.96	0.91	0.09	25.07	38.8%	25.42 \pm 0.69	0.75	0.22	32.26	35.8%	
Mip-NeRF		29.69 \pm 1.24	0.94	0.06	34.79	0.0%	25.99 \pm 0.74	0.78	0.18	43.82	0.0%	
nerfacto [†]												
1	f_1	25.67 \pm 0.30	0.91	0.10	9.90	83.3%	24.46 \pm 0.79	0.81	0.15	11.44	79.5%	
	f_2	26.20 \pm 0.33	0.91	0.09	9.93	82.7%	24.69 \pm 0.75	0.81	0.14	11.44	79.5%	
	f_3	26.17 \pm 0.34	0.91	0.09	9.92	83.1%	24.13 \pm 0.70	0.80	0.16	11.46	79.3%	
2	f_1	26.24 \pm 0.26	0.91	0.09	14.03	29.4%	24.21 \pm 0.86	0.80	0.16	15.90	29.2%	
	f_2	26.56 \pm 0.24	0.91	0.09	14.04	29.3%	24.83 \pm 0.83	0.81	0.14	15.90	29.1%	
	f_3	26.31 \pm 0.22	0.91	0.09	14.04	29.3%	23.51 \pm 0.64	0.78	0.18	15.90	29.1%	
3	f_1	26.16 \pm 0.33	0.91	0.09	18.15	0.0%	23.21 \pm 0.58	0.78	0.19	20.39	0.7%	
	f_2	25.58 \pm 0.27	0.91	0.10	18.18	-0.1%	23.78 \pm 0.79	0.79	0.17	20.39	0.7%	
	f_3	24.79 \pm 0.27	0.90	0.10	18.17	-0.1%	23.04 \pm 0.66	0.77	0.22	20.43	0.5%	
nerfacto [†]		27.61 \pm 0.31	0.93	0.07	18.15	0.00%	25.35 \pm 0.82	0.83	0.12	20.54	0.0%	

first calculating mean results per scene, then averaging all results with equal weight, ensuring that each scene has equal influence. We report per-scene results in the supplementary material.

5.1 Speeding up NeRF and Mip-NeRF

For our experiments, we use the NeRF and Mip-NeRF implementations from Nerfstudio [27]. For the Blender dataset, we choose the default configuration and a batch size of 512 during optimization. For the LLFF dataset, we train for half the epochs compared to Blender, in our case 5×10^5 . We sample in NDC coordinates as done in NeRF [15] and implement a custom LLFF dataloader, modified from the nerf-pytorch [36] codebase. We set the background color to *last_sample*, which results in a modified version of alpha blending.

For the NeRF pipeline, we could either record the activations from Θ_{coarse} or Θ_{fine} . Small ablation studies showed that using activations from Θ_{coarse} results in better performance, hence we used these activations for our experiments.

To attain our weight estimates, we extract the activation features \mathbf{f}_ℓ and apply a function f_i to transform this to a density estimate $\hat{\mathbf{d}}$ for each ray \mathbf{r} . We normalize our approximated density $\hat{\mathbf{d}}$ such that $\mathbf{1}^T \hat{\mathbf{d}} = 1$ and obtain the weights as in [15]. With the resulting weight vector $\hat{\mathbf{w}}$, we perform inverse transform sampling to get our fine samples.



Figure 4: Qualitative evaluation of our approach. For each dataset and architecture, we show test set examples from the coarse-to-fine pipeline and the best rendering using our method. We provide PSNR for each rendered image. For some views, we are able to **outperform** the baseline at lower inference times.

5.2 Speeding up Proposal Networks

For our experiments with proposal network samplers, we use the implementation of the nerfacto model from Nerfstudio [27]. This architecture unifies recent NeRF-related research [3, 13, 16, 29, 32] into a single, universally-applicable method. For our proposal network sampler, we replace the HashMLPDensityField and use the same configuration as in [3], a 4-layer ReLU-MLP with 256 hidden units. We use positional encoding as in NeRF [15]. For a full per-dataset breakdown of hyperparameters, we refer the reader to the supplementary material. We further denote our method as nerfacto[†].

Obtaining our estimated weights \hat{w} works as described above. We note that the nerfacto[†] model uses $\{256, 96\}$ samples for the first and second rounds of proposal sampling and 48 samples for the final shading network evaluation, hence we are able to boost performance significantly by effectively

reducing the size of the first proposal network. For our experiments, we approximate the first round of proposal sampling only: Early experiments showed that our approach does not produce satisfactory results when applied in later stages. We note that for the *leaves* scene, our approach performed worse compared to all other scenes. For this specific scene, the intricate geometry necessitated changes in the training details discussed further in the supplementary material.

5.3 Results

We report quantitative results in Tab. 2 and show our visual results in Fig. 4. For NeRF, the performance gain is not as significant as for Mip-NeRF. This is due to the different number of coarse samples N_c : For Mip-NeRF, $N_c = 128$, contrary to $N_c = 64$ for NeRF, thus we are able to increase the performance significantly by skipping almost the entire coarse pass of the network. We also see that as we increase ℓ , the speedup compared to the baseline method decreases. We find that Eqn. (10) in for $\ell = 2$ works well across the board. All run-time results for NeRF and Mip-NeRF were obtained on an NVIDIA Quadro RTX 8000, using a batch size of 1024 during evaluation.

For the nerfacto[†] model, we observe a huge performance gain, especially for $\ell = 1$. As the first pass through Θ_{prop} uses $N_s = 256$ samples, skipping this step results in a lot less computation, which is exacerbated due to the low overhead required for the shading pass. Again, we find that Eqn. (10) applied in layer $\ell = 2$ performed best across various scenes. For $\ell = 3$, we observe no performance gain, as we effectively skip one single layer and introduce some overhead with our method. All run-time results for nerfacto[†] were obtained on an NVIDIA RTX 2070 Super using a batch size of 1024 during evaluation.

We further note that with our approach, we can also record the activation at a lower resolution and then perform upsampling to get a full-resolution activation for use with our proposed method. This results in a substantial improvement in performance, which we discuss further in the supplementary material. We also experimented with more samples for the shading pass to compensate for a faster sampling pass, which improved visual and quantitative performance for NeRF, Mip-NeRF and nerfacto[†]. We provide small ablation studies for this in the supplementary.

5.4 Failure Cases

For some test set views, using Eqn. (9) results in rays \mathbf{r} with $\hat{\mathbf{w}} = \mathbf{0}$. This produces completely wrong sample placement with inverse transform sampling, leading to a wrong color prediction for this ray \mathbf{r} . We highlight these cases in Tab. 2. These failures are especially evident for Mip-NeRF, as the fine pass does not include the uniformly distributed samples. This is a further reason for the use of Eqns. (10) and (11).

6 Limitations

Although our approach achieves good results for a variety of architectures and for real-world and synthetic datasets, it is not clear whether our approach works for hybrid or explicit representations such as [5, 16, 25, 35]. In general, recent NeRF-related research moves more towards these representations, as they are much easier to accelerate. For our method, the use of other activation functions such as SIREN [23] could alter the behaviour of the intermediate features we investigated. We also use handcrafted functions to obtain density estimates from intermediate activations: In the future, these functions could be learned from data to lead to even better results. Our approach also requires pre-trained architectures and careful analysis to acquire good results.

7 Conclusion and Outlook

We have presented a generally applicable approach to analyze the intermediate activations of deep, coordinate-based ReLU-MLPs. On top, we proposed a method to obtain density estimates using these deep features. We improve performance significantly while mostly preserving the high quality of the trained model, without any fine-tuning or changes to the underlying network architecture. There exist several exciting future research directions for our method beyond our investigated context of neural rendering, and we believe our concepts could be applied to any other task involving coordinate-based MLPs.

References

- [1] Relja Arandjelović and Andrew Zisserman. NeRF in detail: Learning to sample for view synthesis. *arXiv CoRR*, abs/2106.05264, 2021.
- [2] 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. In *ICCV*, 2021.
- [3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *CVPR*, 2022.
- [4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial Radiance Fields. In *ECCV*, 2022.
- [5] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance Fields without Neural Networks. In *CVPR*, 2022.
- [6] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-Fidelity Neural Rendering at 200FPS. In *ICCV*, 2021.
- [7] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul E. Debevec. Baking Neural Radiance Fields for Real-Time View Synthesis. In *ICCV*, 2021.
- [8] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. EfficientNeRF: Efficient Neural Radiance Fields. In *CVPR*, 2022.
- [9] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy J. Mitra. ReLU Fields: The Little Non-linearity That Could. In *SIGGRAPH*, 2022.
- [10] Jonas Kulhanek and Torsten Sattler. Tetra-NeRF: Representing Neural Radiance Fields Using Tetrahedra. *arXiv CoRR*, abs/2304.09987, 2023.
- [11] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. AdaNeRF: Adaptive Sampling for Real-time Rendering of Neural Radiance Fields. In *ECCV*, 2022.
- [12] Haotong Lin, Sida Peng, Zhen Xu, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Efficient Neural Radiance Fields with Learned Depth-Guided Sampling. In *SIGGRAPH Asia*, 2022.
- [13] 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.
- [14] 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 TOG*, 38(4):1–14, 2019.
- [15] 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.
- [16] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM TOG*, 41(4):1–15, 2022.
- [17] 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. *Comput. Graph. Forum*, 40(4):45–59, 2021.
- [18] Michael Oechsle, Songyou Peng, and Andreas Geiger. UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In *ICCV*, 2021.
- [19] Martin Piala and Ronald Clark. TermiNeRF: Ray Termination Prediction for Efficient Neural Rendering. In *3DV*, 2021.
- [20] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the Spectral Bias of Neural Networks. In *ICLR*, 2019.
- [21] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang M. Yi, and Andrea Tagliasacchi. DeRF: Decomposed Radiance Fields. In *CVPR*, 2021.
- [22] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *ICCV*, 2021.
- [23] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions. In *NeurIPS*, 2020.
- [24] Jost T. Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. In *ICLR*, 2015.
- [25] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. In *CVPR*, 2022.
- [26] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. In *NeurIPS*, 2020.
- [27] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa.

- Nerfstudio: A Modular Framework for Neural Radiance Field Development. *arXiv CoRR*, abs/2302.04264, 2023.
- [28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *NeurIPS*, 2017.
 - [29] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields. In *CVPR*, 2022.
 - [30] Ziyu Wan, Christian Richardt, Aljaž Božič, Chao Li, Vijay Rengarajan, Seonghyeon Nam, Xiaoyu Xiang, Tuotuo Li, Bo Zhu, Rakesh Ranjan, and Jing Liao. Learning Neural Duplex Radiance Fields for Real-Time View Synthesis. In *CVPR*, 2023.
 - [31] Zhou Wang, Alam C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE TIP*, 13(4):600–612, 2004.
 - [32] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF—: Neural Radiance Fields Without Known Camera Parameters. *arXiv CoRR*, abs/2102.07064, 2021.
 - [33] Liwen Wu, Jae Yong Lee, Anand Bhattacharjee, Yuxiong Wang, and David Forsyth. DIVeR: Real-time and Accurate Neural Radiance Fields with Deterministic Integration for Volume Rendering. In *CVPR*, 2022.
 - [34] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural Fields in Visual Computing and Beyond. *Comput. Graph. Forum*, 41(2):641–676, 2022.
 - [35] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based Neural Radiance Fields. In *CVPR*, 2022.
 - [36] Lin Yen-Chen. NeRF-pytorch. <https://github.com/yenchenlin/nerf-pytorch/>, 2020.
 - [37] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *ECCV*, 2014.
 - [38] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*, 2018.

A Additional Training Details

We provide additional training details for nerfacto[†] across both used datasets.

A.1 Blender Dataset

For the Blender [15] dataset, we set $t_n = 2.0$, $t_f = 6.0$ for all scenes. We disable the camera optimizer, the appearance embedding, the distortion loss and the scene contraction. We manually set the background color to white. For all scenes, we use a batch size of 2048 rays during training and use a uniform initial proposal network sampler. We train nerfacto[†] for 16.5×10^3 epochs, as in [27].

A.2 LLFF Dataset

For the LLFF [14] dataset, we disable the camera optimizer, the appearance embedding and the distortion loss. We use a ℓ_∞ scene contraction as default for nerfacto [27], contrary to the use of NDC coordinates in NeRF and Mip-NeRF. We do not use our custom LLFF dataparser, we instead opt for the nerfstudio dataparser, which requires pre-processing the data¹. This further shows that our approach also works in warped space. For all scenes, we use a batch size of 1024 rays during training and a batch size of 2048 during evaluation. We train nerfacto[†] for 30×10^3 epochs, as in [27].

For the *leaves* scenes, we use a uniform initial proposal network sampler, as the piece-wise uniform sampler did not result in stable optimization. We note that for this scene, the results are below our obtained average, with a noticeable drop in all metrics.

B Ablation Studies

Here, we present two extensions to our method, which improve visual quality or lower inference times further. For each dataset, we use our method with $\ell = 1$, f_2 and compare against our improvements in Tab. 3. We perform our experiments for our variants of NeRF [15] and Mip-NeRF [2].

B.1 Upsampling Activations

As our approach builds on the intermediate activations for a proposal of density along a ray, a natural extension to our method is to obtain the activations $\mathbf{A}^{(\ell)}$ at a lower resolution. To this end, we perform an ablation study, where we record $\mathbf{A}^{(\ell)}$ at a resolution of $\frac{w}{2} \times \frac{w}{2} \times \frac{N_c}{2} \times N_h$, then perform upsampling to obtain a full-resolution activation $\mathbf{A}^{(\ell)}$. As our interpolation method, we choose nearest neighbor upsampling due to its negligible cost. Similarly to this approach, DONeRF [17] filters along the depth axis and across neighboring rays to obtain a smoothed depth target for their oracle network.

B.2 Better Visual Quality

As we reduce the inference cost at equal number of fine samples N_f , we perform an ablation study with increased N_f , compensating for the achieved speedup. For each model, we choose to add $\frac{N_c}{2}$ samples, which roughly compensates for the speedup when using $\ell = 1$, f_2 .

C Further Motivation for our Approach

To gain further intuition for our approach, we provide a visual example using real-world data. We take a trained Mip-NeRF and evaluate it on a random view of the *chair* scene. We record the activations $\mathbf{A}^{(\ell)}$ for a single test view, as well as the densities σ for each ray. To gain the most insight from a single example, we choose a ray \mathbf{r} for which

$$\mathbf{r} \in \arg \max_{\mathbf{r} \in \mathcal{R}} \sum_{N_s} \sigma_i. \quad (12)$$

¹https://docs.nerf.studio/en/latest/quickstart/custom_dataset.html

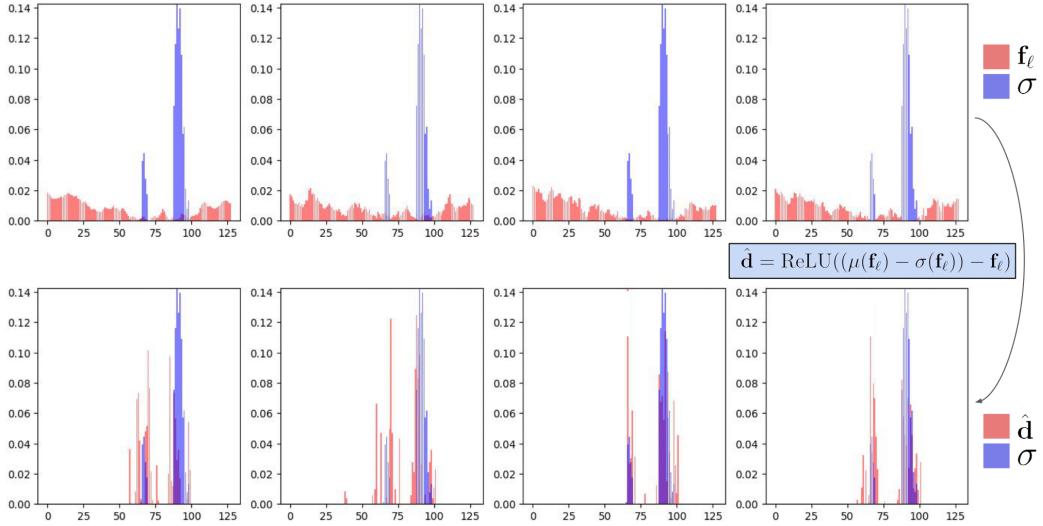


Figure 5: Our approach visualized on a real-world example. Using one of our proposed functions, we can obtain a density estimate sufficiently close to the estimate provided by the uniform samples.

Table 3: We can either improve the performance of our implementation by obtaining the activations at a lower resolution (\star) or gain visual quality by allocating more fine samples, compensating for the speedup (\ddagger).

NeRF								
ℓ	f	Version	Blender, 800 × 800			LLFF, 1008 × 756		
			PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
2	f_2	\ddagger	29.05 \pm 1.14	0.93	0.07	25.12 \pm 0.62	0.74	0.24
		\star	29.48 \pm 1.14	0.93	0.07	25.25 \pm 0.64	0.75	0.23
		NeRF	28.90 \pm 1.12	0.93	0.08	24.60 \pm 0.57	0.73	0.26
Mip-NeRF								
2	f_2	\ddagger	29.04 \pm 1.17	0.93	0.07	25.44 \pm 0.70	0.75	0.22
		\star	29.67 \pm 1.23	0.94	0.06	25.71 \pm 0.72	0.77	0.19
		Mip-NeRF	28.94 \pm 1.16	0.93	0.08	24.90 \pm 0.64	0.74	0.25

We normalize the densities and the activation features such that $\sigma_i \in [0, 1]$, $\mathbf{f}_\ell \in [0, 1]$ (element-wise). In Fig. 5, we visualize the relative scale of the activations with respect to the densities for MLP layers $\ell \in \{1, \dots, 4\}$. We observe the same behavior as described previously: Minima along a ray in activation feature space indicate regions of high density.



Figure 6: Visual artefacts appear when the distribution along a ray does not permit a well-behaved density estimate.

D Failure Cases

We note that for some example views, the distribution of activations along some rays does not permit a reasonable density estimate using our hand-crafted functions. In Fig. 6, we showcase the visual effects of these failure cases. This error only occurs when using f_1 and can be mitigated by a more conservative approach for using the density, hence the use of f_2, f_3 .

Another interesting detail is that we observe this behavior only for the Blender scenes *hotdog*, *lego*, *ship* and the LLFF scene *T-Rex* given $\ell \in \{2, 3\}$, with most of these failures happening when $\ell = 3$. The quantitative results in Tabs. 4 and 5 show that these scenarios occur most often with Mip-NeRF.

Contrasting Mip-NeRF’s coarse-to-fine sampling approach with NeRF’s, the reason for these results is the inclusion of the N_c uniformly distributed samples for the fine network pass in NeRF. Mip-NeRF does not include the original samples and is thus more prone to faulty density prediction. When looking at the Blender dataset, we additionally see that errors for this dataset are quantitatively and visually overpenalized due to the transparent background.

We did not find a satisfactory explanation for this change in distribution at the time of writing, hence we encourage future work in this direction. We believe this could be an important insight for designing future view synthesis architectures.

E Additional Results

In Tabs. 4 and 5, we show per-scene results for the Blender dataset [15] and the LLFF dataset [14]. For the *drums* scene, we manage to outperform Mip-NeRF quantitatively over all $n = 200$ test set images using our proposed densities.

Table 4: Results for the Blender dataset [15] for our experiments, per scene. All results are given as mean \pm standard deviation for all $n = 200$ images of the test set. We color code **best**, second-best and third-best for each scene. In addition, we highlight **failures** of our method.

NeRF									
ℓ	Fct.	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
1	f_1	30.10 ± 1.67	22.33 ± 0.88	26.21 ± 1.05	32.41 ± 1.58	26.79 ± 1.18	28.05 ± 1.04	29.10 ± 1.31	24.86 ± 0.83
	f_2	30.12 ± 1.71	23.30 ± 1.02	28.25 ± 0.65	33.40 ± 1.88	28.99 ± 0.92	28.58 ± 1.06	30.53 ± 0.85	26.30 ± 0.69
	f_3	30.43 ± 1.61	23.23 ± 1.01	28.14 ± 0.70	33.26 ± 1.82	28.75 ± 0.88	28.61 ± 1.03	30.48 ± 0.91	26.16 ± 0.71
2	f_1	30.66 ± 1.65	23.57 ± 1.14	25.83 ± 2.79	33.17 ± 2.09	28.54 ± 2.29	28.87 ± 1.07	31.27 ± 1.25	22.04 ± 1.54
	f_2	30.19 ± 1.75	23.63 ± 1.08	28.45 ± 0.64	33.61 ± 1.97	30.03 ± 1.03	28.73 ± 1.10	30.97 ± 0.73	26.79 ± 0.78
	f_3	30.69 ± 1.71	23.91 ± 1.15	29.03 ± 0.67	33.86 ± 2.07	30.62 ± 0.94	29.01 ± 1.10	31.37 ± 0.83	27.03 ± 0.83
3	f_1	27.93 ± 2.92	20.26 ± 1.68	21.74 ± 1.89	28.10 ± 3.54	20.56 ± 2.71	26.13 ± 2.51	28.15 ± 3.01	20.48 ± 1.30
	f_2	28.98 ± 2.19	20.46 ± 0.96	22.42 ± 1.04	32.97 ± 2.02	27.62 ± 2.13	26.18 ± 1.45	25.28 ± 0.98	25.91 ± 0.82
	f_3	27.93 ± 3.04	19.62 ± 0.73	21.29 ± 0.79	32.15 ± 2.86	26.14 ± 3.37	24.17 ± 1.46	23.59 ± 0.65	25.71 ± 1.11
NeRF		31.16 ± 1.69	24.09 ± 1.20	29.26 ± 0.77	34.33 ± 2.23	31.34 ± 1.09	29.08 ± 1.08	31.75 ± 0.90	27.92 ± 1.02
Mip-NeRF									
1	f_1	30.14 ± 1.69	22.33 ± 0.85	26.10 ± 0.95	33.06 ± 3.36	27.52 ± 1.04	28.21 ± 1.07	28.54 ± 1.00	24.44 ± 1.65
	f_2	30.01 ± 1.76	23.36 ± 1.10	27.60 ± 0.80	33.64 ± 1.76	29.83 ± 1.04	28.60 ± 1.16	30.25 ± 0.73	26.69 ± 0.81
	f_3	30.34 ± 1.69	23.34 ± 1.11	27.44 ± 0.85	33.87 ± 1.86	29.70 ± 0.94	28.61 ± 1.14	29.93 ± 0.79	26.56 ± 0.84
2	f_1	30.77 ± 1.55	23.91 ± 1.21	27.46 ± 1.39	31.67 ± 3.64	28.97 ± 2.01	27.97 ± 1.46	30.99 ± 1.07	11.65 ± 2.22
	f_2	30.13 ± 1.78	23.66 ± 1.12	28.07 ± 0.72	33.57 ± 1.70	30.38 ± 1.12	28.62 ± 1.22	30.97 ± 0.80	26.89 ± 0.92
	f_3	30.65 ± 1.71	23.95 ± 1.18	28.39 ± 0.78	33.72 ± 1.76	30.82 ± 1.11	28.83 ± 1.21	31.30 ± 0.85	27.16 ± 0.93
3	f_1	28.77 ± 4.21	22.31 ± 3.77	25.96 ± 3.44	17.52 ± 8.20	23.23 ± 6.21	27.85 ± 3.85	30.90 ± 2.57	11.20 ± 3.34
	f_2	29.19 ± 2.13	22.68 ± 1.13	25.16 ± 1.19	33.60 ± 1.77	29.64 ± 1.55	27.75 ± 1.83	25.06 ± 1.72	26.66 ± 0.99
	f_3	28.07 ± 3.22	21.86 ± 1.59	23.22 ± 1.91	33.44 ± 1.81	29.64 ± 2.34	26.27 ± 2.91	21.98 ± 0.97	27.10 ± 0.91
Mip-NeRF		31.13 ± 1.61	23.82 ± 1.20	28.52 ± 0.80	34.41 ± 2.01	31.07 ± 1.24	29.31 ± 1.23	31.54 ± 0.85	27.71 ± 0.97
nerfacto [†]									
1	f_1	28.30 ± 1.55	20.80 ± 0.89	22.06 ± 1.22	31.48 ± 1.80	25.89 ± 1.03	24.19 ± 1.37	27.49 ± 0.89	25.14 ± 1.16
	f_2	29.07 ± 1.64	21.17 ± 0.98	22.36 ± 1.28	31.98 ± 1.99	26.89 ± 1.24	24.35 ± 1.37	28.19 ± 0.87	25.56 ± 1.18
	f_3	29.07 ± 1.66	21.18 ± 1.00	22.21 ± 1.28	32.03 ± 2.00	26.91 ± 1.19	24.30 ± 1.36	28.02 ± 0.89	25.66 ± 1.21
2	f_1	29.16 ± 1.71	20.89 ± 0.93	22.08 ± 1.23	31.19 ± 1.76	27.26 ± 1.19	24.61 ± 1.34	29.18 ± 1.25	25.57 ± 1.18
	f_2	29.35 ± 1.71	21.36 ± 1.06	22.17 ± 1.23	31.46 ± 1.83	28.16 ± 1.30	24.56 ± 1.34	29.45 ± 1.33	25.93 ± 1.25
	f_3	28.56 ± 1.59	21.24 ± 1.02	22.04 ± 1.22	31.18 ± 1.76	27.40 ± 1.17	24.66 ± 1.33	29.51 ± 1.38	25.92 ± 1.25
3	f_1	27.96 ± 1.54	20.59 ± 0.83	22.02 ± 1.21	31.00 ± 1.71	28.50 ± 1.91	24.64 ± 1.34	29.52 ± 1.59	25.04 ± 1.12
	f_2	28.01 ± 1.57	20.64 ± 0.84	22.00 ± 1.21	31.03 ± 1.71	25.24 ± 1.64	24.36 ± 1.31	28.16 ± 1.32	25.16 ± 1.13
	f_3	27.95 ± 1.54	20.60 ± 0.84	22.02 ± 1.21	31.01 ± 1.71	22.69 ± 1.00	23.53 ± 1.27	25.55 ± 1.07	24.97 ± 1.12
nerfacto [†]		30.64 ± 1.90	22.11 ± 1.39	22.86 ± 1.41	32.67 ± 2.32	30.65 ± 1.78	24.76 ± 1.34	29.78 ± 1.46	27.38 ± 1.67

Table 5: Results for the LLFF dataset [14] for our experiments, per scene. All results are given as mean \pm standard deviation for each n test set images for each scene. We color code **best**, **second-best** and **third-best** for each scene. In addition, we highlight **failures** of our method.

NeRF									
ℓ	Fct.	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	T-Rex
1	f_1	23.83 \pm 0.01	27.61 \pm 0.35	28.12 \pm 1.36	25.11 \pm 0.91	21.38 \pm 0.41	19.35 \pm 0.25	30.32 \pm 0.41	25.16 \pm 1.26
	f_2	24.19 \pm 0.08	28.03 \pm 0.35	28.27 \pm 1.36	25.52 \pm 0.97	21.35 \pm 0.40	19.57 \pm 0.24	30.81 \pm 0.52	25.44 \pm 1.32
	f_3	24.07 \pm 0.06	27.97 \pm 0.36	28.22 \pm 1.36	25.42 \pm 0.96	21.38 \pm 0.40	19.48 \pm 0.24	30.73 \pm 0.52	25.43 \pm 1.32
2	f_1	23.76 \pm 0.01	27.68 \pm 0.38	27.95 \pm 1.29	24.84 \pm 0.91	21.08 \pm 0.42	19.48 \pm 0.26	28.66 \pm 0.21	24.32 \pm 1.10
	f_2	24.03 \pm 0.08	27.95 \pm 0.37	28.17 \pm 1.31	25.26 \pm 0.95	21.23 \pm 0.41	19.62 \pm 0.25	29.44 \pm 0.35	25.30 \pm 1.28
	f_3	23.92 \pm 0.06	27.86 \pm 0.36	28.09 \pm 1.31	25.14 \pm 0.95	21.19 \pm 0.41	19.56 \pm 0.26	29.13 \pm 0.28	25.21 \pm 1.25
3	f_1	23.61 \pm 0.01	27.65 \pm 0.38	27.74 \pm 1.26	24.34 \pm 0.82	20.97 \pm 0.42	19.50 \pm 0.26	28.18 \pm 0.13	21.73 \pm 0.65
	f_2	23.92 \pm 0.07	27.95 \pm 0.34	28.06 \pm 1.28	24.96 \pm 0.91	21.17 \pm 0.41	19.65 \pm 0.24	28.94 \pm 0.23	25.26 \pm 1.27
	f_3	23.82 \pm 0.05	27.88 \pm 0.33	27.95 \pm 1.26	24.79 \pm 0.89	21.12 \pm 0.41	19.61 \pm 0.25	28.66 \pm 0.17	25.17 \pm 1.25
NeRF		24.70 \pm 0.15	28.61 \pm 0.40	29.07 \pm 1.49	26.05 \pm 1.07	21.56 \pm 0.41	19.95 \pm 0.23	31.49 \pm 0.74	25.72 \pm 1.38
Mip-NeRF									
1	f_1	23.56 \pm 0.31	27.73 \pm 0.26	28.04 \pm 1.28	24.79 \pm 0.92	21.38 \pm 0.31	19.34 \pm 0.21	30.57 \pm 0.63	25.16 \pm 1.37
	f_2	23.93 \pm 0.33	28.26 \pm 0.24	28.50 \pm 1.34	25.51 \pm 1.07	21.37 \pm 0.31	19.59 \pm 0.19	30.89 \pm 0.69	25.56 \pm 1.45
	f_3	23.80 \pm 0.32	28.16 \pm 0.24	28.32 \pm 1.32	25.35 \pm 1.03	21.39 \pm 0.31	19.50 \pm 0.20	30.79 \pm 0.68	25.46 \pm 1.43
2	f_1	23.41 \pm 0.25	27.88 \pm 0.42	28.13 \pm 1.26	24.89 \pm 1.01	21.23 \pm 0.33	19.41 \pm 0.22	30.47 \pm 0.62	24.72 \pm 1.32
	f_2	23.72 \pm 0.29	28.33 \pm 0.25	28.52 \pm 1.27	25.49 \pm 1.10	21.34 \pm 0.32	19.60 \pm 0.22	30.92 \pm 0.69	25.58 \pm 1.44
	f_3	23.60 \pm 0.29	28.29 \pm 0.25	28.37 \pm 1.27	25.35 \pm 1.08	21.34 \pm 0.33	19.53 \pm 0.22	30.78 \pm 0.65	25.51 \pm 1.43
3	f_1	23.51 \pm 0.30	25.69 \pm 0.68	28.31 \pm 1.29	23.68 \pm 1.40	20.74 \pm 0.10	18.75 \pm 0.19	30.14 \pm 0.67	15.30 \pm 0.94
	f_2	23.83 \pm 0.32	28.37 \pm 0.26	28.72 \pm 1.25	25.52 \pm 1.08	21.32 \pm 0.33	19.67 \pm 0.20	30.90 \pm 0.68	25.61 \pm 1.45
	f_3	23.71 \pm 0.30	28.34 \pm 0.26	28.60 \pm 1.26	25.37 \pm 1.06	21.31 \pm 0.33	19.61 \pm 0.20	30.81 \pm 0.65	25.57 \pm 1.44
Mip-NeRF		24.54 \pm 0.37	28.69 \pm 0.26	29.47 \pm 1.29	26.13 \pm 1.16	21.59 \pm 0.30	20.01 \pm 0.16	31.59 \pm 0.87	25.92 \pm 1.51
nerfacto [†]									
1	f_1	23.52 \pm 0.07	26.60 \pm 1.18	27.59 \pm 1.32	25.18 \pm 1.45	17.98 \pm 0.21	19.03 \pm 0.52	29.81 \pm 0.56	25.95 \pm 1.01
	f_2	23.72 \pm 0.09	26.78 \pm 1.17	27.70 \pm 1.29	26.44 \pm 1.34	17.86 \pm 0.05	19.06 \pm 0.51	30.09 \pm 0.49	25.88 \pm 1.07
	f_3	23.10 \pm 0.03	26.30 \pm 1.15	27.22 \pm 1.27	24.88 \pm 1.11	18.14 \pm 0.12	18.92 \pm 0.57	29.45 \pm 0.48	25.06 \pm 0.86
2	f_1	22.32 \pm 0.33	26.06 \pm 1.41	26.84 \pm 1.04	25.97 \pm 1.30	18.61 \pm 0.26	18.57 \pm 0.64	30.20 \pm 0.51	25.07 \pm 1.43
	f_2	22.98 \pm 0.25	26.64 \pm 1.42	27.53 \pm 1.18	26.65 \pm 1.26	19.25 \pm 0.33	18.99 \pm 0.54	30.28 \pm 0.54	26.34 \pm 1.12
	f_3	21.97 \pm 0.21	25.21 \pm 1.06	26.45 \pm 1.00	23.85 \pm 0.82	18.29 \pm 0.02	18.33 \pm 0.54	29.98 \pm 0.50	24.03 \pm 1.04
3	f_1	22.44 \pm 0.14	25.83 \pm 1.01	27.23 \pm 1.20	23.65 \pm 0.65	18.34 \pm 0.01	17.88 \pm 0.61	28.03 \pm 0.26	22.28 \pm 0.74
	f_2	22.06 \pm 0.48	25.96 \pm 1.49	26.76 \pm 1.38	24.58 \pm 0.83	18.46 \pm 0.12	17.77 \pm 0.24	28.81 \pm 0.53	25.87 \pm 1.27
	f_3	22.13 \pm 0.26	25.77 \pm 1.14	27.05 \pm 1.26	23.51 \pm 0.78	18.32 \pm 0.03	17.40 \pm 0.59	27.62 \pm 0.25	22.49 \pm 0.97
nerfacto [†]		23.83 \pm 0.15	27.01 \pm 1.31	27.94 \pm 1.18	26.91 \pm 1.44	21.28 \pm 0.35	19.11 \pm 0.51	30.36 \pm 0.53	26.39 \pm 1.11