Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields Supplemental Material

Jonathan T. Barron[1] Ben Mildenhall[1] Matthew Tancik[2]
Peter Hedman[1] Ricardo Martin-Brualla[1] Pratul P. Srinivasan[1]
[1]Google [2]UC Berkeley

1. Conical Frustum Integral Derivations

In order to derive formulas for the various moments of the uniform distribution over a conical frustum, we consider an axis-aligned cone parameterized as $(x, y, z) = \varphi(r, t, \theta) = (rt\cos\theta, rt\sin\theta, t)$ for $\theta \in [0, 2\pi)$, $t \geq 0$, $|r| \leq \dot{r}$. This change of variables from Cartesian space gives us a differential term:

$$dx\,dy\,dz = |\det(D\varphi)(r,t,\theta)|dr\,dt\,d\theta \tag{1}$$

$$= \begin{vmatrix} t\cos\theta & t\sin\theta & 0 \\ r\cos\theta & r\sin\theta & 1 \\ -rt\sin\theta & rt\cos\theta & 0 \end{vmatrix} dr\,dt\,d\theta \tag{2}$$

$$= (rt^2\cos^2\theta + rt^2\sin\theta)dr\,dt\,d\theta \tag{3}$$

$$= rt^2 dr\,dt\,d\theta\,. \tag{4}$$

The volume of the conical frustum (which serves as the normalizing constant for the uniform distribution) is:

$$V = \int_0^{2\pi}\int_{t_0}^{t_1}\int_0^{\dot{r}} rt^2\,dr\,dt\,d\theta \tag{5}$$

$$= \frac{\dot{r}^2}{2}\cdot\frac{t_1^3 - t_0^3}{3}\cdot 2\pi \tag{6}$$

$$= \pi\dot{r}^2\frac{t_1^3 - t_0^3}{3} \tag{7}$$

Thus the probability density function for points uniformly sampled from the conical frustum is rt^2/V. The first moment of t is:

$$E[t] = \frac{1}{V}\int_0^{2\pi}\int_{t_0}^{t_1}\int_0^{\dot{r}} t\cdot rt^2\,dr\,dt\,d\theta \tag{8}$$

$$= \frac{1}{V}\int_0^{2\pi}\int_{t_0}^{t_1}\int_0^{\dot{r}} rt^3\,dr\,dt\,d\theta \tag{9}$$

$$= \frac{1}{V}\cdot\pi\dot{r}^2\frac{t_1^4 - t_0^4}{4} \tag{10}$$

$$= \frac{3(t_1^4 - t_0^4)}{4(t_1^3 - t_0^3)}\,. \tag{11}$$

The moments of x and y are both zero by symmetry. The second moment of t is

$$E[t^2] = \frac{1}{V}\int_0^{2\pi}\int_{t_0}^{t_1}\int_0^{\dot{r}} t^2\cdot rt^2\,dr\,dt\,d\theta \tag{12}$$

$$= \frac{1}{V}\int_0^{2\pi}\int_{t_0}^{t_1}\int_0^{\dot{r}} rt^4\,dr\,dt\,d\theta \tag{13}$$

$$= \frac{1}{V}\cdot\pi\dot{r}^2\frac{t_1^5 - t_0^5}{5} \tag{14}$$

$$= \frac{3(t_1^5 - t_0^5)}{5(t_1^3 - t_0^3)}\,. \tag{15}$$

And the second moment of x is:

$$E[x^2] = \frac{1}{V}\int_0^{2\pi}\int_{t_0}^{t_1}\int_0^{\dot{r}} (rt\cos\theta)^2\cdot rt^2\,dr\,dt\,d\theta \tag{16}$$

$$= \frac{1}{V}\int_{t_0}^{t_1}\int_0^{\dot{r}} r^3t^4\int_0^{2\pi}\cos^2\theta\,d\theta\,dr\,dt \tag{17}$$

$$= \frac{1}{V}\cdot\frac{\dot{r}^4}{4}\cdot\frac{t_1^5 - t_0^5}{5}\cdot\pi \tag{18}$$

$$= \frac{\dot{r}^2}{4}\cdot\frac{3(t_1^5 - t_0^5)}{5(t_1^3 - t_0^3)}\,. \tag{19}$$

The second moment of y is the same by symmetry. All cross terms in the covariance are z, also by symmetry.

With these moments defined, we can construct the mean and covariance for a random point within our conical frustum. The mean along the ray direction μ_t is simply the first moment with respect to t:

$$\mu_t = \frac{3(t_1^4 - t_0^4)}{4(t_1^3 - t_0^3)}\,. \tag{20}$$

The variance of the conical frustum with respect to t follows from the definition of variance as $\mathrm{Var}(t) = \mathrm{E}[t^2] - \mathrm{E}[t]^2$:

$$\sigma_t^2 = \frac{3(t_1^5 - t_0^5)}{5(t_1^3 - t_0^3)} - \mu_t^2\,. \tag{21}$$

The variance of the conical frustum with respect to its radius r is equal to the variance of the frustum with respect to x or

(by symmetry) y. Since the first moment with respect to x is zero, the variance is equal to the second moment:

$$\sigma_r^2 = \dot{r}^2 \left(\frac{3(t_1^5 - t_0^5)}{20(t_1^3 - t_0^3)} \right). \tag{22}$$

Computing all three of these quantities in their given form is numerically unstable — the ratio of the differences between t_1 and t_0 raised to large powers is difficult to compute accurately when t_0 and t_1 are near each other, which occurs frequently during training. Using these quantities in practice often produces 0 or NaN instead of accurate values, which causes training to fail. We therefore reparameterize these equations as a function of the center and spread of t_0 and t_1: $t_\mu = (t_0 + t_1)/2$, $t_\delta = (t_1 - t_0)/2$. This allows us to rewrite each mean and variance as a first-order term that is then corrected by higher-order terms, which are scaled by t_δ. This gives us stable and accurate values even when t_δ is small. Our reparameterized values are:

$$\mu_t = t_\mu + \frac{2t_\mu t_\delta^2}{3t_\mu^2 + t_\delta^2}, \quad \sigma_t^2 = \frac{t_\delta^2}{3} - \frac{4t_\delta^4(12t_\mu^2 - t_\delta^2)}{15(3t_\mu^2 + t_\delta^2)^2},$$
$$\sigma_r^2 = \dot{r}^2 \left(\frac{t_\mu^2}{4} + \frac{5t_\delta^2}{12} - \frac{4t_\delta^4}{15(3t_\mu^2 + t_\delta^2)} \right). \tag{23}$$

Note that our multivariate Gaussian approximation of a conical frustum will be inaccurate if there is a significant difference between the base and top radii of the frustum, which will be true for frustums that are very near the camera's center of projection when the camera FOV is large. This is highly uncommon in most datasets, but may be an issue if one were to use mip-NeRF in unusual circumstances, such as macro photography with a fisheye lens.

2. The L Hyperparameter in PE and IPE

IPE features can be viewed as a generalization of PE features: $\gamma(\mathbf{x}) = \gamma(\boldsymbol{\mu} = \mathbf{x}, \boldsymbol{\Sigma} = \mathbf{0})$. Or more rigorously, PE features can be thought of as "hard" IPE features in which all points are assumed to have identical isotropic covariance matrices whose variance has been heuristically determined by the L hyperparameter: the value of L determines the frequency at which PE features are truncated, just as the Gaussian function of variance in IPE serves as a "soft" truncation of IPE features. Because the "soft" maximum frequency of IPE features is determined entirely by the geometry and intrinsics of the camera, IPE features do not depend on the L hyperparameter, and so using IPE features removes the need for tuning L. This is because in PE the L parameter determines where the high frequencies in the PE are truncated, but in IPE those high frequencies are naturally attenuated by the size of the multivariate Gaussian used as input to the encoding: the smaller the Gaussian, the more high frequencies will be retained. To demonstrate this, we performed as

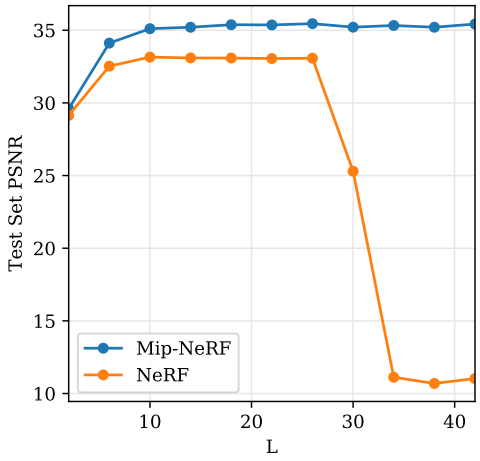

Figure 1: PSNRs for NeRF and mip-NeRF on the test set of the *lego* scene, as we vary the positional encoding degree L. In NeRF, performance decreases due to overfitting for large values of L, but in mip-NeRF this parameter is effectively removed from tuning — it can just be set to a large value and forgotten, because IPE features "tune" their own frequencies automatically.

"sweep" of L in both mip-NeRF and NeRF, and report the test-set PSNR for a single scene, which is visualized in Figure 1. We see that in NeRF, there is a range of values for L in which performance is maximized, but values that are too large or too small will hurt performance. But in mip-NeRF, we see that L can be set to an arbitrarily large value and performance is unaffected. In practice, in all mip-NeRF experiments in the paper we set $L = 16$, which is a value that results in the last dimension of all IPE features constructed during training to be less than numerical epsilon.

3. Hyperparameters

In all experiments in the paper we take care to use exactly the same set of hyperparameters that were used in Mildenhall *et al.* [6], so as to isolate the specific contributions of mip-NeRF as they relate to cone-casting and IPE features. The three relevant hyperparameters that govern mip-NeRF's behavior are: 1) the number of samples N drawn at each of the two levels ($N = 128$), 2) the histogram "padding" hyperparameter α on the coarse transmittance weights that are used to sample the fine t values ($\alpha = 0.01$), and 3) the multiplier λ on the "coarse" component of the loss function ($\lambda = 0.1$). And though mip-NeRF adds these three hyperparameters, it also deprecates three NeRF hyperparameters that are no longer used: 1) The number of samples N_c drawn for the "coarse" MLP ($N_c = 64$), 2) The number of samples N_f drawn for the "fine" MLP ($N_f = 128$), and 3) The degree L used for the spatial positional encoding ($L = 10$). The α parame-

ter used by mip-NeRF serves a similar purpose as the balance between N_c and N_f did in NeRF — a larger value of α biases the final samples used during rendering towards a uniform distribution, just as a larger value of N_c biases the final samples (which are the sorted union of the uniform coarse samples and the biased fine samples) towards a uniform distribution. Mip-NeRF's multiplier λ has no analog in NeRF, as NeRF's usage of two distinct MLPs means that the "coarse" and "fine" losses in NeRF do not need to be balanced — thankfully, though mip-NeRF adds the need to tune this new hyperparameter λ, it simultaneously removes the need to tune the L hyperparameter as discussed in Section 2, so the total number of hyperparameters that require tuning remains constant across the two models.

Before running the experiments in the paper, we briefly tuned the α and λ hyperparameters by hand on the validation set of the *lego* scene. N was not tuned, and was just set to 128 such that the total number of MLP evaluations used by mip-NeRF matched the total number used by NeRF.

4. Forward-Facing Scenes

Note that this paper does not evaluate on the LLFF dataset [5], which consists of scenes captured by a "forward-facing" handheld cellphone camera. For these scenes, NeRF trained and evaluated models in a "normalized device coordinates" (NDC) space. NDC coordinates work by nonlinearly warping a frustum-shaped space into a unit cube, which sidesteps some otherwise challenging design decisions (such as how an unbounded 3D space should be represented using positional encoding). NDC coordinates can only be used for these "forward-facing" scenes; in scenes where the camera rotates significantly (which is the case for the vast majority of 3D datasets) NeRF uses conventional 3D "world coordinates". One interesting consequence of NDC space is that the 3D volume corresponding to a pixel is *not* a frustum, but is instead a rectangle — in NDC the spatial support of a pixel in the xy plane *does not* increase with the distance from the image plane, as it would in conventional projective geometry.

We briefly experimented with a variant of mip-NeRF that works in NDC space by casting *cylinders* instead of cones. The average PSNR achieved by JaxNeRF on this task is 26.843, and this cylinder-casting variant of mip-NeRF achieves an average PSNR of 26.838. Because this mip-NeRF variant roughly matches the accuracy of NeRF, the only substantial benefit it appears to provide is removing the need to tune the L parameter in positional encoding. This result provides some insight into why NeRF works so well on forward-facing scenes: in NDC space there is little difference between NeRF's "incorrect" aliased approach of casting rays and tuning the L hyperparameter (which as discussed in Section 2, is approximately equivalent to using IPE features with isotropic Gaussians) and the more "correct" anti-aliased approach of mip-NeRF. In essence, NeRF is already able to get most of the benefit provided by cone-casting and IPE features in NDC space, because in NDC space NeRF's aliased model is already very similar to mip-NeRF's approach. This interplay between scene parameterization and anti-aliasing suggests that a signal processing analysis of coordinate spaces in neural rendering problems may provide additional unexpected benefits or insights.

5. Model Details

The primary contributions of this paper are the use of cone tracing, integrated positional encoding features, and our use of a single unified multiscale model (as opposed to NeRF's separate per-scale models), which together allow mip-NeRF to better handle multiscale data and reduce aliasing. Additionally, mip-NeRF includes a small number of changes that do not meaningfully change mip-NeRF's accuracy or speed, but slightly simplify our method and increase its robustness during optimization. These "miscellaneous" changes, as noted by the "w/o Misc." ablation in the main paper, do not significantly affect mip-NeRF's performance, but are described here in full for the sake of reproducibility with the hopes that future work will find them useful.

5.1. Identity Concatenation

In the original NeRF paper, the input to the MLP is not just the positional encoding of the position and view direction, but is instead the concatenation of the positional encoding with the position and view direction being encoded. We found this "identity" encoding to not contribute meaningfully to performance or speed, and its presence makes the formalization of our IPE features somewhat challenging, so this in mip-NeRF this identity mapping is removed and the only input to the MLP is the integrated positional encoding itself.

5.2. Activation Functions

In the original NeRF paper, the activation functions used by the MLP to construct the predicted density τ and color $\mathbf{c}$ are a ReLU and a sigmoid, respectively. Instead of a ReLU as the activation function to produce τ, we use a shifted softplus: $\log(1 + \exp(x - 1))$. We found that using a softplus yielded a smoother optimization problem that is less prone to catastrophic failure modes in which the MLP emits negative values everywhere (in which case all gradients from τ are zero and optimization will fail). The shift by -1 within the softplus is equivalent to initializing the biases that produce τ in mip-NeRF to -1, and this causes initial τ values to be small. Initializing the density of the NeRF to small values results in slightly faster optimization at the beginning of training, as dense scene content causes gradients from scene content "behind" that dense content to be suppressed. Instead of a sigmoid to produce color $\mathbf{c}$, we use a "widened"

sigmoid that saturates slightly outside of $[0, 1]$ (the range of input RGB intensities): $(1 + 2\epsilon)/(1 + \exp(-x)) - \epsilon$, with $\epsilon = 0.001$. This avoids an uncommon failure mode in which training tries to explain away a black or white pixel by saturating network activations into the tails of the sigmoid where the gradient is zero, which may cause optimization to fail. By having the network saturate at values slightly outside of the range of input values, activations are never encouraged to saturate. These changes to activation functions have little effect on performance, but we found that they improved training stability when using large learning rates (though all results in this paper use the same lower learning rates used by Mildenhall *et al*. [6] for fair comparison).

5.3. Optimization

In all experiments we train mip-NeRF and JaxNeRF using the default training procedure specified in the JaxNeRF codebase: 1 million iterations of Adam [2] with a batch size of 4096 and a learning rate that is annealed logarithmically from $\eta_0 = 5 \cdot 10^{-4}$ to $\eta_n = 5 \cdot 10^{-6}$. We additionally "warm up" the learning rate using the functionality provided by JaxNeRF, which does not improve the performance of mip-NeRF itself, but which we found to improve the stability of some of the mip-NeRF ablations. To allow our ablations to be competitive, and to enable a fair comparison across all models, we therefore use this warm up strategy in all mip-NeRF and JaxNeRF experiments. Because the warm up procedure in JaxNeRF is not described in its documentation [1], for the sake of reproducibility we will describe it here. For the first $n_w = 2500$ iterations of optimization, we scale the basic learning rate by an additional scale factor that is smoothly annealed between $\lambda_w = 0.01$ and 1 during this warm up period. The learning rate at iteration i during training is:

$$\begin{aligned}\eta_i =&(\lambda_w + (1 - \lambda_w) \sin((\pi/2) \operatorname{clip}(i/n_w, 0, 1))) \\ &\times (\exp((1 - i/n) \log(\eta_0) + (i/n) \log(\eta_n))) \end{aligned} \quad (24)$$

See Figure 2 for a visualization.

5.4. View Dependent Effects

We handle viewing directions exactly as was done in NeRF: the ray direction $\mathbf{d}$ is normalized, positionally encoded ($L = 4$), and injected into the last layer of the MLP after τ is predicted but before $\mathbf{c}$ is predicted. This is omitted from our notation in the main paper for simplicity's sake. see Mildenhall *et al*. for details [6].

6. Supersampling Baseline

In the main paper we presented a generous baseline approach in which NeRF is trained on only full-resolution images (thereby sidestepping its poor performance when trained on multi-resolution data) and then evaluated on our multiscale Blender dataset by brute-force supersampling: rendering a full-resolution image that is then downsampled to match the resolution of the ground truth. This roughly matches the performance of mip-NeRF, but is 22× slower and relies on "oracle" scale information that does not exist for most datasets. Here we explore an alternative supersampling baseline, in which we train an extension of NeRF on the multiscale dataset while supersampling during *both* training and evaluation: for every pixel we cast multiple jittered rays (sampled uniformly at random) through the spatial footprint of each pixel, render each ray with the NeRF, and then use the mean of those rendered values as the predicted color of that pixel in the loss function. As shown by the results of this experiment (Table 1) this brute-force supersampling model not only performs worse than mip-NeRF even when casting as many as 16 rays per pixel, but is also significantly more expensive during both training and evaluation.

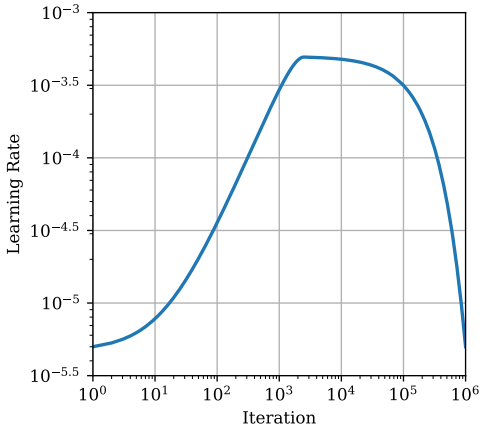

Figure 2: The learning rate used in all JaxNeRF and mip-NeRF experiments.

7. Alternative Gaussian Positional Encoding

During experimentation we explored alternative approaches for featurizing the mean and covariance matrix of the multivariate Gaussians used by mip-NeRF. One such alternative strategy is to simply apply positional encoding to the mean and to the (signed) square root of the elements of the covariance matrix, and use the concatenation of the two as input. Specifically, we compute the positional encoding of $\boldsymbol{\mu}$ with $L = 12$, and compute the positional encoding of $\operatorname{vec}(\operatorname{triu}(\operatorname{sign}(\boldsymbol{\Sigma}) \circ \sqrt{|\boldsymbol{\Sigma}|}))$ with $L = 2$. We found that this approach performs comparably to the IPE features presented in the main paper, as shown in Table 2. We chose to advocate for IPE features in the main paper instead of this concatenation alternative because 1) IPE features are more compact (thereby reducing model size and evaluation time), 2) IPE features are easy to justify and reason about (as they

	PSNR ↑				SSIM ↑				LPIPS ↓					Train Time	Test Time	
	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res.	Full Res.	1/2 Res.	1/4 Res.	1/8 Res	Avg. ↓	(hours)	(sec/MP)	# Params
NeRF + Area, Center, 1× SS	27.471	28.016	27.816	26.657	0.9187	0.9301	0.9365	0.9304	0.1064	0.0924	0.0934	0.1064	0.0362	2.85	2.61	1,191K
NeRF + Area, Center, 4× SS	28.424	29.420	29.863	29.233	0.9297	0.9426	0.9526	0.9547	0.0807	0.0598	0.0530	0.0536	0.0259	17.69	10.44	1,191K
NeRF + Area, Center, 16× SS	31.566	33.116	33.982	32.933	0.9524	0.9660	0.9753	0.9768	0.0537	0.0316	0.0227	0.0216	0.0144	37.18	41.76	1,191K
Mip-NeRF	32.629	34.336	35.471	35.602	0.9579	0.9703	0.9786	0.9833	0.0469	0.0260	0.0168	0.0120	0.0114	2.79	2.48	612K

Table 1: Here we evaluate mip-NeRF against an extension of NeRF in which brute-force supersampling with jittered rays is used during training and evaluation, on our multiscale Blender dataset ("16× SS" indicates that 16 rays are cast per pixel, etc). Mip-NeRF is able to outperform this baseline by a significant margin in terms of quality, while also being 13× faster to train and 16× faster to evaluate.

Multiscale Blender	PSNR ↑	SSIM ↑	LPIPS ↓	Avg. ↓
Integrated PE	**34.51**	**0.973**	**0.025**	**0.0113**
Concatenated PE	34.40	**0.973**	**0.025**	0.0114

Blender	PSNR ↑	SSIM ↑	LPIPS ↓	Avg. ↓
Integrated PE	**33.09**	**0.961**	0.043	0.0161
Concatenated PE	**33.09**	**0.961**	**0.042**	**0.0160**

Table 2: An evaluation of the IPE features against an alternative approach in which the mean and covariance of the multivariate Gaussian corresponding to a conical frustum are positionally encoded and concatenated. Both approaches perform comparably on the multiscale and single-scale Blender datasets.

approximate an expectation of positional encoding features with respect to a conical frustum), and 3) IPE features have no hyperparameters (while this concatenation alternative is sensitive to its two L hyperparameters and the design decisions used when parameterizing $\mathbf{\Sigma}$).

This experiment with using this alternative to IPE also provides some insight into the inner workings of mip-NeRF. While IPE features are insensitive to the off-diagonal elements of $\mathbf{\Sigma}$, this concatenation alternative should endow the MLP with the ability to reason about the correlation of dimensions of the multivariate Gaussian. The fact that this ability does not improve accuracy may suggest that correlation is not a helpful cue, which contradicted the intuition of the authors. Additionally, this experiment reinforces the assertions made in the paper that the reason for mip-NeRF's improved performance is its explicit modeling of conical frustums, as opposed to NeRF's usage of point samples along a ray. Though it is critical that the geometry of image formation be modeled accurately, there are likely many effective ways to featurize that geometry.

8. Additional Results

Multiscale Blender Dataset. To demonstrate the relative accuracy of mip-NeRF compared to NeRF on each individual scene in the multiscale Blender dataset, the error metrics for each individual scene are provided in Table 3. Mip-NeRF yields a significant reduction in error compared to NeRF across all scenes. Renderings produced by mip-NeRF and baseline algorithms compared to the ground truth can be visually inspected in Figures 3 and 4.

Blender Dataset. Test-set error metrics for each individual scene in the (single scale) Blender dataset of Mildenhall *et al*. [6] can be seen in Table 4. Mip-NeRF yields lower error rates than NeRF on all scenes and all metrics.

References

[1] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an efficient JAX implementation of NeRF, 2020. `http://github.com/google-research/google-research/tree/master/jaxnerf`.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

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

[4] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *SIGGRAPH*, 2019.

[5] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima K. Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *SIGGRAPH*, 2019.

[6] 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. *ECCV*, 2020.

[7] Vincent Sitzmann, Michael Zollhoefer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. *NeurIPS*, 2019.

	Average PSNR							
	chair	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*
NeRF (Jax Implementation) [1, 6]	29.923	23.273	27.153	32.001	27.748	26.295	28.401	26.462
NeRF + Area Loss	30.277	24.032	27.149	32.025	27.602	26.533	28.120	26.834
NeRF + Area, Centered Pixels	33.460	25.802	30.400	35.672	31.606	30.155	32.633	30.019
NeRF + Area, Center, Misc.	33.394	25.874	30.369	35.641	31.646	30.184	32.601	30.092
Mip-NeRF	37.141	27.021	33.188	39.313	35.736	32.558	38.036	33.083
Mip-NeRF w/o Misc.	37.275	26.979	33.160	39.357	35.749	32.563	37.997	33.078
Mip-NeRF w/o Single MLP	37.310	26.922	33.045	39.378	35.605	32.635	38.016	33.011
Mip-NeRF w/o Area Loss	35.188	26.063	32.542	37.165	34.319	31.004	35.922	31.636
Mip-NeRF w/o IPE	33.559	25.864	30.499	35.793	31.728	30.272	32.736	30.276

	Average SSIM							
	chair	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*
NeRF (Jax Implementation) [1, 6]	0.9436	0.8908	0.9423	0.9586	0.9256	0.9335	0.9580	0.8607
NeRF + Area Loss	0.9488	0.9028	0.9429	0.9622	0.9274	0.9372	0.9592	0.8610
NeRF + Area, Centered Pixels	0.9710	0.9310	0.9705	0.9794	0.9643	0.9670	0.9800	0.8994
NeRF + Area, Center, Misc.	0.9707	0.9318	0.9705	0.9793	0.9646	0.9671	0.9799	0.9004
Mip-NeRF	0.9875	0.9450	0.9836	0.9880	0.9843	0.9767	0.9928	0.9221
Mip-NeRF w/o Misc.	0.9877	0.9448	0.9835	0.9880	0.9842	0.9767	0.9927	0.9227
Mip-NeRF w/o Single MLP	0.9875	0.9432	0.9829	0.9876	0.9836	0.9763	0.9922	0.9211
Mip-NeRF w/o Area Loss	0.9817	0.9371	0.9823	0.9849	0.9792	0.9731	0.9911	0.9175
Mip-NeRF w/o IPE	0.9714	0.9322	0.9713	0.9796	0.9658	0.9678	0.9804	0.9039

	Average LPIPS							
	chair	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*
NeRF (Jax Implementation) [1, 6]	0.0347	0.0689	0.0324	0.0279	0.0410	0.0452	0.0307	0.0948
NeRF + Area Loss	0.0414	0.0762	0.0438	0.0365	0.0568	0.0499	0.0444	0.1139
NeRF + Area, Centered Pixels	0.0281	0.0593	0.0264	0.0240	0.0348	0.0330	0.0249	0.0865
NeRF + Area, Center, Misc.	0.0283	0.0586	0.0264	0.0241	0.0346	0.0330	0.0249	0.0850
Mip-NeRF	0.0111	0.0439	0.0135	0.0121	0.0127	0.0186	0.0065	0.0624
Mip-NeRF w/o Misc.	0.0111	0.0436	0.0136	0.0123	0.0127	0.0186	0.0066	0.0620
Mip-NeRF w/o Single MLP	0.0113	0.0443	0.0142	0.0122	0.0132	0.0187	0.0068	0.0628
Mip-NeRF w/o Area Loss	0.0171	0.0503	0.0146	0.0151	0.0163	0.0259	0.0095	0.0665
Mip-NeRF w/o IPE	0.0276	0.0578	0.0259	0.0240	0.0340	0.0320	0.0231	0.0829

Table 3: Per-scene results on the test set images of the multiscale Blender dataset presented in this work. We report the arithmetic mean of each metric averaged over the four scales used in the dataset.

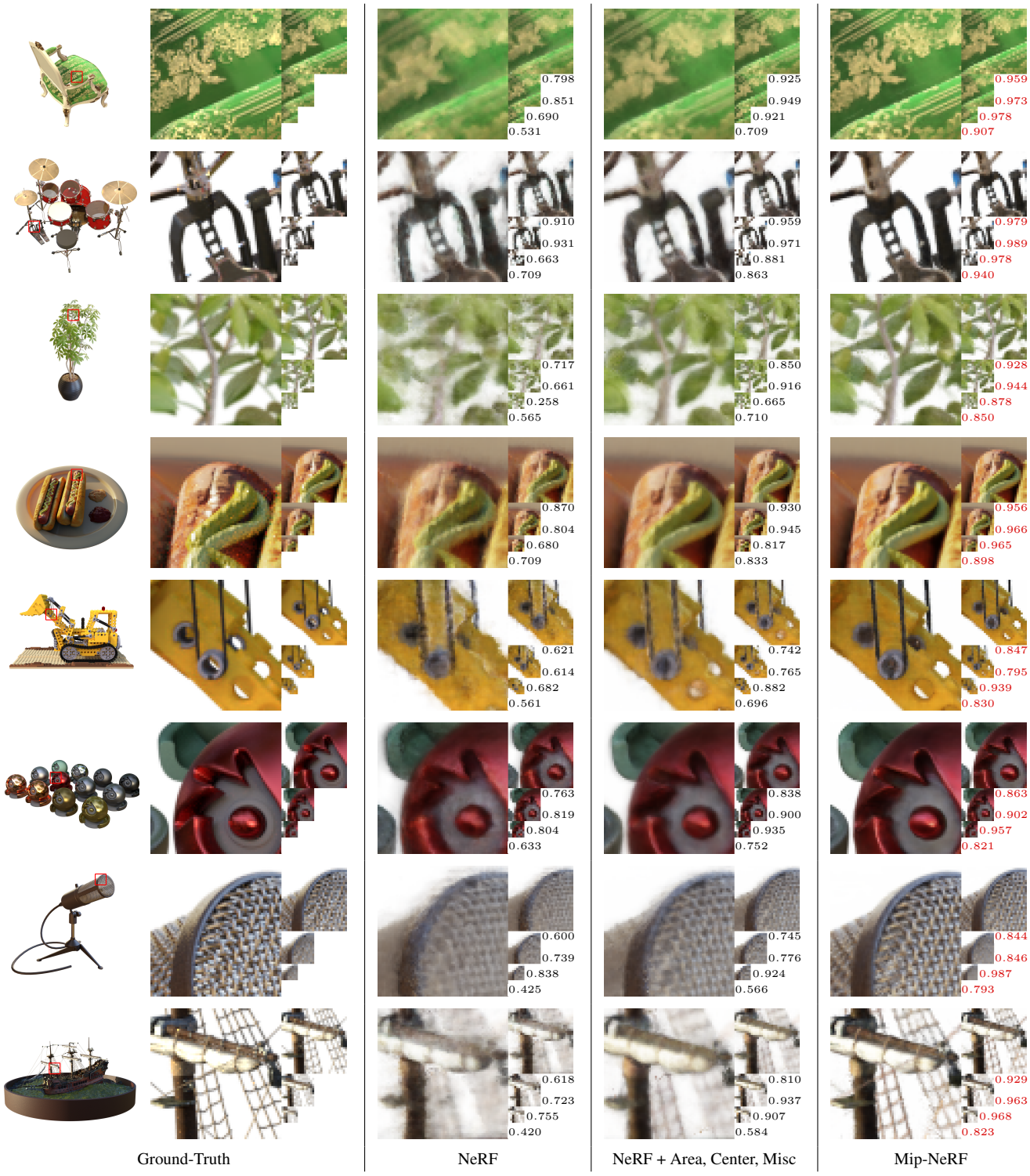

Figure 3: Visualizations of the output renderings from mip-NeRF compared to the ground truth, NeRF, and our improved version of NeRF, on test set images from the 8 scenes in our multiscale Blender dataset. We visualize a cropped region of each scene for better visualization, and render out that scene at 4 different resolutions, displayed as an image pyramid. The SSIM for each scale of each image pyramid truth is shown to its lower right, with the highest SSIM for each algorithm at each scale highlighted in red.

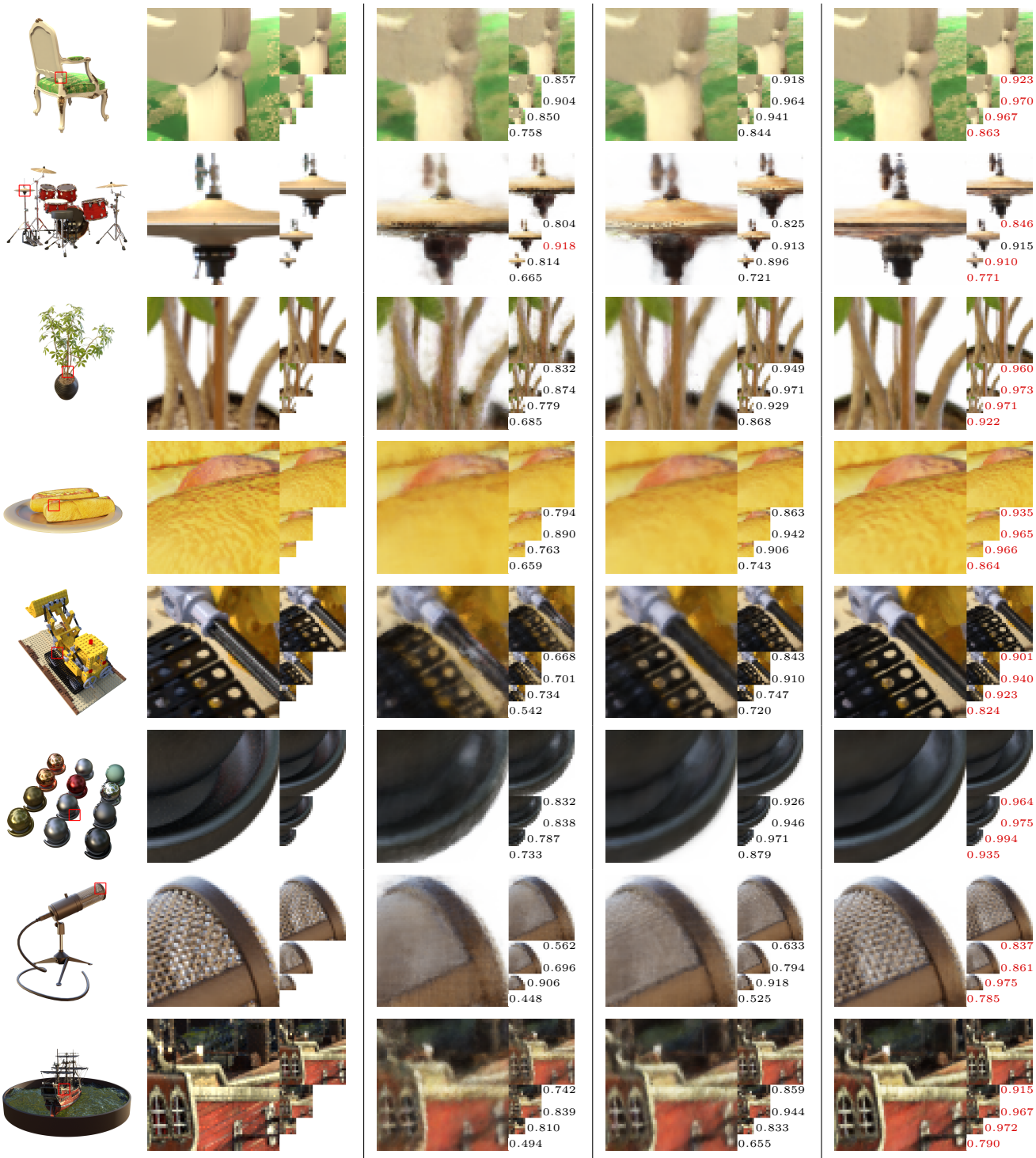

Figure 4: Additional visualizations of the output renderings from mip-NeRF compared to the ground truth, NeRF, and an improved version of NeRF presented in this work, on test set images from the 8 scenes in our multiscale Blender dataset, in the same format as Figure 3.

	PSNR							
	chair	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*
SRN [7]	26.96	17.18	20.73	26.81	20.85	18.09	26.85	20.60
Neural Volumes [4]	28.33	22.58	24.79	30.71	26.08	24.22	27.78	23.93
LLFF [5]	28.72	21.13	21.79	31.41	24.54	20.72	27.48	23.22
NSVF [3]	33.19	25.18	31.23	37.14	32.29	32.68	34.27	27.93
NeRF (TF Implementation) [6]	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65
NeRF (Jax Implementation) [1, 6]	34.17	25.08	30.39	36.82	33.31	30.03	34.78	29.30
NeRF + Centered Pixels	34.88	25.17	31.02	37.13	34.39	30.50	35.38	29.95
NeRF + Center, Misc.	34.94	25.19	31.05	37.15	34.12	30.47	35.33	29.95
Mip-NeRF	35.14	25.48	33.29	37.48	35.70	30.71	36.51	30.41
Mip-NeRF w/o Single MLP	35.07	25.28	32.52	37.34	34.93	30.38	35.59	30.55
Mip-NeRF w/o Misc.	35.16	25.46	32.96	37.55	35.68	30.69	36.32	30.47
Mip-NeRF w/o IPE	35.10	25.23	31.30	37.17	34.89	30.56	35.75	29.85
Mip-NeRF, Stopped Early	34.21	25.23	30.79	36.89	33.72	29.86	35.02	29.44

	SSIM							
	chair	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*
SRN [7]	0.910	0.766	0.849	0.923	0.809	0.808	0.947	0.757
Neural Volumes [4]	0.916	0.873	0.910	0.944	0.880	0.888	0.946	0.784
LLFF [5]	0.948	0.890	0.896	0.965	0.911	0.890	0.964	0.823
NSVF [3]	0.968	0.931	0.973	0.980	0.960	0.973	0.987	0.854
NeRF (TF Implementation) [6]	0.967	0.925	0.964	0.974	0.961	0.949	0.980	0.856
NeRF (Jax Implementation) [1, 6]	0.975	0.925	0.967	0.979	0.968	0.953	0.987	0.869
NeRF + Centered Pixels	0.979	0.928	0.971	0.980	0.973	0.956	0.989	0.877
NeRF + Center, Misc.	0.979	0.927	0.971	0.980	0.972	0.956	0.989	0.877
Mip-NeRF	0.981	0.932	0.980	0.982	0.978	0.959	0.991	0.882
Mip-NeRF w/o Single MLP	0.980	0.929	0.977	0.981	0.976	0.956	0.989	0.883
Mip-NeRF w/o Misc.	0.981	0.932	0.979	0.982	0.978	0.959	0.991	0.883
Mip-NeRF w/o IPE	0.981	0.929	0.972	0.981	0.975	0.958	0.990	0.878
Mip-NeRF, Stopped Early	0.976	0.927	0.969	0.979	0.969	0.954	0.988	0.869

	LPIPS							
	chair	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*
SRN [7]	0.106	0.267	0.149	0.100	0.200	0.174	0.063	0.299
Neural Volumes [4]	0.109	0.214	0.162	0.109	0.175	0.130	0.107	0.276
LLFF [5]	0.064	0.126	0.130	0.061	0.110	0.117	0.084	0.218
NSVF [3]	0.043	0.069	0.017	0.025	0.029	0.021	0.010	0.162
NeRF (TF Implementation) [6]	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206
NeRF (Jax Implementation) [1, 6]	0.026	0.071	0.032	0.030	0.031	0.047	0.012	0.150
NeRF + Centered Pixels	0.022	0.069	0.028	0.028	0.026	0.043	0.010	0.143
NeRF + Center, Misc.	0.022	0.069	0.028	0.028	0.027	0.044	0.011	0.142
Mip-NeRF	0.021	0.065	0.020	0.027	0.021	0.040	0.009	0.138
Mip-NeRF w/o Single MLP	0.022	0.067	0.023	0.028	0.024	0.044	0.011	0.135
Mip-NeRF w/o Misc.	0.021	0.066	0.022	0.026	0.021	0.040	0.009	0.136
Mip-NeRF w/o IPE	0.020	0.068	0.027	0.028	0.024	0.041	0.009	0.142
Mip-NeRF, Stopped Early	0.027	0.074	0.035	0.031	0.035	0.046	0.013	0.155

Table 4: Per-scene results on the test set images of the (single-scale) Blender dataset from Mildenhall *et al*. [6]