Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields Supplemental Material

Jonathan T. Barron Ben Mildenhall Dor Verbin Pratul P. Srinivasan Peter Hedman
Google Research

1. Video Results

This supplement includes video results for scenes from the 360 benchmark. We also present video results for new more-challenging scenes that we have captured to qualitatively demonstrate various kinds of aliasing and our model's ability to ameliorate that aliasing.

Affine Generative Latent Optimization The video results presented in mip-NeRF 360 use the appearance-embedding approach of NeRF-W [8], which assigns short GLO vectors [4] to each image in the training set that are concatenated to the input of the view-dependent MLP. By jointly optimizing over these embeddings during training, optimization is able to explain away view-dependent effects such as variation in exposure and lighting. On scenes with significant illumination variation we found this approach to be reasonably effective, but it is limited by the relatively small size of the view-dependent branch of our MLP compared to NeRF-W and mip-NeRF 360. We therefore use an approach similar to NeRF-W's GLO approach, but instead of optimizing over a short feature that is concatenated onto our bottleneck vector, we optimize over an affine transformation of our bottleneck vector itself. Furthermore, we express this affine transformation not just with a per-image latent embedding, but also with a small MLP that maps from a per-image latent embedding to an affine transformation. We found that optimizing over the large space of embeddings and MLP weights resulted in faster training than the GLO baseline, which allows the model to more easily explain per-image appearance variation without placing floaters in front of training cameras.

We allocate an 128-length vector for each image in the training set and use those vectors as input to a two-layer MLP with 128 hidden units whose output is two vectors (scale and shift) that are the same length as the bottleneck. The internal activation of the MLP is a ReLU, and the final activation that yields our scaling is $\exp$. We scale and shift each bottleneck by the two MLP outputs before evaluating the view-dependent MLP. This approach of optimizing an affine function of an internal activation resembles prior techniques in the literature for modulating activations to control "style" and appearance [6, 12, 13]. This technique is only used to produce our video results and is not used for the experiments mentioned in the paper, as it does not improve quantitative performance on our metrics.

2. Multisampling Pattern Derivation

The hexagonal multisampling pattern presented in the paper was constructed so as to satisfy several criteria:

1. Samples should be uniformly distributed along the ray, to ensure good coverage along the ray.
2. Samples should be uniformly distributed in terms of angles around the ray.
3. The sample mean and covariance of the set of samples should match the analytical mean and covariance of the conical frustum.
4. The number of points should be as small as possible, for the sake of efficiency.

We additionally chose to always distribute samples at a distance from the ray that is proportional to the radius of the cone at whatever t value the sample is located. This simplifies the analysis of our pattern, though it does mean that none of our samples are placed exactly along the ray, which may be contrary to the reader's expectations. Experimentally, we found little value in adding additional multisamples exactly along the ray, which is consistent with the performance of the unscented transform baseline (which places multiple points along the ray).

Before constructing our conical frustum-shaped multisampling pattern, we can simplify our analysis by first constructing an n-point multisampling pattern for a cylinder. Here is a set of n coordinates that, for a carefully chosen $\boldsymbol{\theta}$ and n, has a mean of $\vec{0}$ and a covariance of I_3:

$$\left\{ \begin{bmatrix} \cos(\theta_j)/\sqrt{3} \\ \sin(\theta_j)/\sqrt{3} \\ \frac{j-(n-1)/2}{\sqrt{n(n^2-1)/12}} \end{bmatrix} \middle| \, j = 0, 1, \ldots, n \right\}. \tag{1}$$

Perhaps surprisingly, for small values of n and assum-

ing uniformly-distributed values of θ, this zero-mean and identity-covariance property appears to only hold for $n = 6$ and two specific choices of $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_1 = [\,0, {}^{2\pi}/_{3}, {}^{4\pi}/_{3}, {}^{3\pi}/_{3}, {}^{5\pi}/_{3}, {}^{\pi}/_{3}\,], \quad (2)$$

$$\boldsymbol{\theta}_2 = [\,0, {}^{3\pi}/_{3}, {}^{2\pi}/_{3}, {}^{5\pi}/_{3}, {}^{4\pi}/_{3}, {}^{\pi}/_{3}\,]. \quad (3)$$

Because our sampling pattern is rotationally symmetric around θ and bilaterally symmetric around the $z = 0$ plane of the cylinder, rotating or mirroring the coordinates corresponding to these two choices of $\boldsymbol{\theta}$ preserves their zero-mean and identity-covariance. We use $\boldsymbol{\theta}_1$ in our work, because $\boldsymbol{\theta}_2$ exhibits potentially-undesirable higher-order correlation between adjacent angles (note that $\boldsymbol{\theta}_2$ consists of three pairs of adjacent angles, while $\boldsymbol{\theta}_1$ has only two adjacent angles that are nearby).

With this cylindrical multisampling pattern that satisfies our requirements, we can then warp these samples into the shape of a conical frustum, while also shifting and scaling the coordinates such that they match the means and covariances derived in mip-NeRF [2]. This yields the formulas shown in Equations 2 and 3 in the main paper. This warping results in a slight mismatch between the sample covariance of our multisample coordinates and the covariance of the frustum: the full covariance matrices are not necessarily identical. However, the variance along the ray and the total variance perpendicular to the ray are both equal to that of the conical frustum, and the mismatch between full covariances goes to zero as $t \gg \dot{r}$ (which is generally the case for real image data, where $\dot{r}$ is usually small).

3. Scale Featurization

Along with features $\mathbf{f}_\ell$ we also average and concatenate a featurized version of $\{\omega_{j,\ell}\}$ for use as input to our MLP:

$$(2 \cdot \mathrm{mean}_j(\omega_{j,\ell}) - 1)\sqrt{V_{init}^2 + \not\nabla(\mathrm{mean}(V_\ell^2))}\,, \quad (4)$$

where $\not\nabla$ is a stop-gradient operator and V_{init} is the magnitude used to initialize each V_ℓ. This feature takes ω_j (shifted and scaled to $[-1, 1]$) and scales it by the standard deviation of the values in V_ℓ (padded slightly using V_{init}, which guards against the case where a V_ℓ's values shrink to zero during training). This scaling ensures that our featurized scales have roughly the same magnitude features as the features themselves, regardless of how the features may grow or shrink during training. The stop-gradient prevents optimization from indirectly modifying this scale-feature by changing the values in V_ℓ.

When computing the downweighting factor ω, for the sake of speed we use an approximation for $\mathrm{erf}(x)$:

$$\mathrm{erf}(x) \approx \mathrm{sign}(x)\sqrt{1 - \exp(-({}^{4}/_{\pi})x^2)}\,. \quad (5)$$

This has no discernible impact on quality and a very marginal impact on performance.

4. Spatial Contraction

Mip-NeRF 360 [3] parameterizes unbounded scenes with bounded coordinates using a spatial contraction:

$$\mathcal{C}(\mathbf{x}) = \begin{cases} \mathbf{x} & \|\mathbf{x}\| \leq 1 \\ \left(2 - \frac{1}{\|\mathbf{x}\|}\right)\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right) & \|\mathbf{x}\| > 1\,. \end{cases} \quad (6)$$

This maps values in $[-\infty, \infty]^d$ to $[-2, 2]^d$ such that resolution in the contracted domain is proportional to what is required by perspective projection — scene content near the origin is allocated significant model capacity, but distant scene content is allocated model capacity that is roughly proportional to disparity (inverse distance).

Given our multisampled isotropic Gaussians $\{\mathbf{x}_j, \sigma_j\}$, we need a way to efficiently apply a scene contraction. Contracting the means of the Gaussians simply requires evaluating each $\mathcal{C}(\mathbf{x}_j)$, but contracting the scale is non-trivial, and the approach provided by mip-NeRF 360 [3] for contracting a *multivariate* Gaussian is needlessly expressive and expensive for our purposes. Instead, we linearize the contraction around each $\mathbf{x}_j$ to produce $\mathbf{J}_\mathcal{C}(\mathbf{x}_j)$, the Jacobian of the contraction at $\mathbf{x}_j$, and we produce an isotropic scale in the contracted space by computing the geometric mean of the eigenvalues of $\mathbf{J}_\mathcal{C}(\mathbf{x}_j)$. This is the same as computing the determinant of the absolute value of $\mathbf{J}_\mathcal{C}(\mathbf{x}_j)$ and taking its dth root ($d = 3$ in our case, as our coordinates are 3D):

$$\mathcal{C}(\sigma_j) = \sigma_j |\det(\mathbf{J}_\mathcal{C}(\mathbf{x}_j))|^{1/d}\,. \quad (7)$$

This is equivalent to using the approach in mip-NeRF 360 [3] to apply a contraction to a multivariate Gaussian with a covariance matrix of $\sigma^2 I_d$ and then identifying the isotropic Gaussian with the same generalized variance as the contracted multivariate Gaussian, but requires significantly less compute. This can be accelerated further by deriving a closed-form solution for our specific contraction:

$$|\det(\mathbf{J}_\mathcal{C}(\mathbf{x}_j))|^{1/3} = \left(\frac{\sqrt[3]{2\max(1, \|\mathbf{x}_j\|) - 1}}{\max(1, \|\mathbf{x}_j\|)}\right)^2 \quad (8)$$

5. Blurring A Step Function

Algorithm 1 contains pseudocode for the algorithm described in the paper for convolving a step function with a rectangular pulse to yield a piecewise linear spline. This code is valid JAX/Numpy code except that we have overloaded `sort()` to include the behavior of `argsort()`.

6. Power Transformation Details

Here we expand upon the power transformation $\mathcal{P}(x, \lambda)$ presented in the paper. First, let's expand upon its definition

Algorithm 1 $\mathbf{x}_r, \mathbf{y}_r = \text{blur\_stepfun}(\mathbf{x}, \mathbf{y}, r)$

$\mathbf{x}_r,\ sortidx = \text{sort}(\text{concatenate}([\mathbf{x} - r, \mathbf{x} + r]))$
$\mathbf{y}' = ([\mathbf{y}, 0] - [0, \mathbf{y}])/(2r)$
$\mathbf{y}'' = \text{concatenate}([\mathbf{y}', -\mathbf{y}'])[sortidx[:-1]]$
$\mathbf{y}_r = [0, \text{cumsum}((\mathbf{x}_r[1:] - \mathbf{x}_r[:-1])\,\text{cumsum}(\mathbf{y}''))]$

to include its two removable singularities and its limits as λ approaches $\pm\infty$:

$$\mathcal{P}(x, \lambda) = \begin{cases} x & \lambda = 1 \\ \log(1 + x) & \lambda = 0 \\ e^x - 1 & \lambda = \infty \\ 1 - e^{-x} & \lambda = -\infty \\ \frac{|\lambda - 1|}{\lambda}\left(\left(\frac{x}{|\lambda - 1|} + 1\right)^{\lambda} - 1\right) & \text{otherwise} \end{cases} \tag{9}$$

Note that the $\lambda = -\infty, 0, +\infty$ cases can and should be implemented using the `log1p()` and `expm1()` operations that are standard in numerical computing and deep learning libraries. As discussed, the slope of this function is 1 near the origin, but further from the origin λ can be tuned to describe a wide family of shapes: exponential, squared, logarithmic, inverse, inverse-square, and (negative) inverse-exponential. Scaling the x input to $\mathcal{P}$ lets us control the effective range of inputs where $\mathcal{P}(x, \lambda)$ is approximately linear. The second derivative of $\mathcal{P}$ at the origin is ± 1, depending on if λ is more or less than 1, and the output of $\mathcal{P}$ is bounded by $\frac{\lambda - 1}{\lambda}$ when $\lambda < 0$.

This power transformation relates to the general robust loss $\rho(x, \alpha, c)$ [1], as ρ can be written as $\mathcal{P}$ applied to squared error: $\rho(x, \lambda, 1) = \mathcal{P}(x^2/2, \lambda/2)$. $\mathcal{P}$ also resembles a reparameterized Yeo–Johnson transformation [15] but altered such that the transform always resembles a straight line near the origin (the second derivative of the Yeo-Johnson transformation at the origin is unbounded).

Distortion Loss As discussed in the paper, our power transformation is used to curve metric distance into a normalized space where resampling and interlevel supervision can be performed effectively. We also use this transformation to curve metric distance within the distortion loss used in mip-NeRF 360. By tuning a transformation to have a steep gradient near the origin that tapers off into something resembling log-distance, we obtain a distortion loss that more aggressively penalizes "floaters" near the camera, which significantly improves the quality of videos rendered from our model. This does not significantly improve test-set metrics on the 360 dataset, as floaters do not tend to contribute much to error metrics computed on still images. In all of our experiments we set our model's multiplier on distortion loss to 0.005. See Figure 1 for a visualization of our curve and the impact it has on distortion loss.

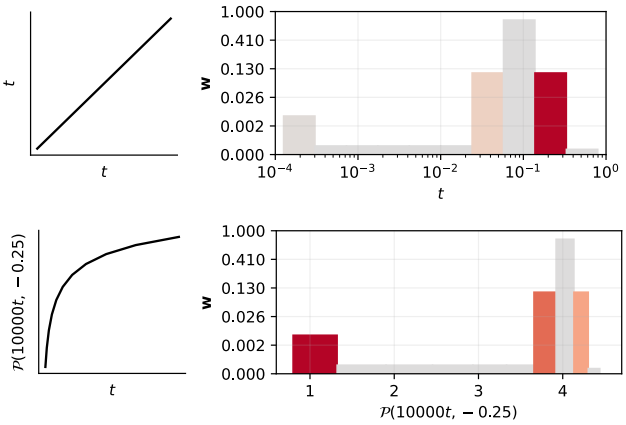

Figure 1: The behavior of mip-NeRF 360's distortion loss can be significantly modified by applying a curve to metric distance, which we do with our power transformation. Top: using a linear curve (*i.e.*, using metric distance t itself) results in a distortion loss that heavily penalizes distant scene content but ignores "floaters" close to the camera (the single large histogram bin near $t = 0$), as can be seen by visualizing the gradient magnitude of distortion loss as a heat-map over the NeRF histogram shown here. Bottom: Curving metric distance with a tuned power transformation $\mathcal{P}(10000x, -0.25))$ before computing distortion loss causes distortion loss to correctly penalize histogram bins near the camera.

7. Model Details

Our model, our "mip-NeRF 360 + iNGP" baseline, and all ablations (unless otherwise stated) were trained for 25k iterations using a batch size of 2^{16}. We use the Adam optimizer [7] with $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-15}$, and we decay our learning rate logarithmically from 10^{-2} to 10^{-3} over training. We use no gradient clipping, and we use an aggressive warm-up for our learning rate: for the first 5k iterations we scale our learning rate by a multiplier that is cosine-decayed from 10^{-8} to 1.

In our iNGP hierarchy of grids and hashes, we use 10 grid scales that are spaced by powers of 2 from 16 to 8192, and we use 4 channels per level. We found this to work slightly better and faster than iNGP's similar approach of spacing scales by $\sqrt{2}$ and having 2 channels per level. Grid sizes n_ℓ that are greater than 128 are parameterized identically to iNGP using a hash of size 128^3.

We inherit the proposal sampling procedure used by mip-NeRF 360: two rounds of proposal sampling where we bound scene geometry and recursively generate new sample intervals, and one final NeRF round in which we render that final set of intervals into an image. We use a distinct NGP and MLP for each round of sampling, as this improves performance slightly. Because high-frequency content is

largely irrelevant for earlier rounds of proposal sampling, we truncate the hierarchy of grids of each proposal NGP at maximum grid sizes of 512 and 2048. We additionally only use a single channel of features for each proposal NGP, as (unlike the NeRF NGP) these models only need to predict density, not density and color.

We found that small view-dependent MLP used by iNGP to be a significant bottleneck to performance, as it limits the model's ability to express and recover complicated view-dependent effects. This not only reduces rendering quality on shiny surfaces, but also causes more "floaters" by encouraging optimization to explain away view-dependent effects with small floaters in front of the camera. We therefore use a larger model: We have a view-dependent bottleneck size of 256, and we process those bottleneck vectors with a 3-layer MLP with 256 hidden units and a skip connection from the bottleneck to the second layer.

Ablations In our non-normalized weight decay ablation, we use weight decay with a multiplier of 10^{-9}, though we found performance to be largely insensitive to what multiplier is used. Our "Naive Supersampling ($6\times$)" and "Jittered Supersampling ($6\times$)" ablations in the paper caused training to run out of memory, so those experiments use half of the batch size used in other experiments and are trained for twice as many iterations.

8. Results

An per-scene version of our single-scale results on the mip-NeRF 360 dataset can be found in Table 1.

We also include per-scene and average results for the Blender dataset [9] in Table 2. For these Blender results, we use the same model as presented in the paper, with the following changes: we set the ray near and far plane distances to 2 and 6 respectively, we use a linear (no-op) curve when spacing sample intervals along each ray, we assume a white background color, and we increase our weight decay multiplier to 10.

References

[1] Jonathan T. Barron. A general and adaptive robust loss function. *CVPR*, 2019.

[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. *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. *CVPR*, 2022.

[4] Piotr Bojanowski, Armand Joulin, David Lopez-Pas, and Arthur Szlam. Optimizing the latent space of generative networks. *ICML*, 2018.

[5] 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`.

[6] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *ICCV*, 2017.

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

[8] 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. *CVPR*, 2021.

[9] 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.

[10] Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, Peter Hedman, Ricardo Martin-Brualla, and Jonathan T. Barron. MultiNeRF: A Code Release for Mip-NeRF 360, Ref-NeRF, and RawNeRF, 2022.

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *SIGGRAPH*, 2022.

[12] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. *CVPR*, 2019.

[13] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. *AAAI*, 2018.

[14] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. BakedSDF: Meshing neural SDFs for real-time view synthesis. *SIGGRAPH*, 2023.

[15] In-Kwon Yeo and Richard A Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 2000.

[16] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492*, 2020.

PSNR	Outdoor					Indoor			
	bicycle	*flowers*	*garden*	*stump*	*treehill*	*room*	*counter*	*kitchen*	*bonsai*
NeRF [9, 5]	21.76	19.40	23.11	21.73	21.28	28.56	25.67	26.31	26.81
mip-NeRF [2]	21.69	19.31	23.16	23.10	21.21	28.73	25.59	26.47	27.13
NeRF++ [16]	22.64	20.31	24.32	24.34	22.20	28.87	26.38	27.80	29.15
Instant NGP [11, 14]	22.79	19.19	25.26	24.80	22.46	30.31	26.21	29.00	31.08
mip-NeRF 360 [3, 10]	24.40	21.64	26.94	26.36	22.81	31.40	29.44	32.02	33.11
mip-NeRF 360 + iNGP	24.51	21.82	27.05	25.08	23.01	31.07	24.01	30.18	31.12
Our Model	25.80	22.40	28.20	27.55	23.89	32.65	29.38	32.50	34.46

SSIM	Outdoor					Indoor			
	bicycle	*flowers*	*garden*	*stump*	*treehill*	*room*	*counter*	*kitchen*	*bonsai*
NeRF [9, 5]	0.455	0.376	0.546	0.453	0.459	0.843	0.775	0.749	0.792
mip-NeRF [2]	0.454	0.373	0.543	0.517	0.466	0.851	0.779	0.745	0.818
NeRF++ [16]	0.526	0.453	0.635	0.594	0.530	0.852	0.802	0.816	0.876
Instant NGP [11, 14]	0.540	0.378	0.709	0.654	0.547	0.893	0.845	0.857	0.924
mip-NeRF 360 [3, 10]	0.693	0.583	0.816	0.746	0.632	0.913	0.895	0.920	0.939
mip-NeRF 360 + iNGP	0.692	0.615	0.840	0.720	0.633	0.911	0.821	0.910	0.930
Our Model	0.769	0.642	0.860	0.800	0.681	0.925	0.902	0.928	0.949

LPIPS	Outdoor					Indoor			
	bicycle	*flowers*	*garden*	*stump*	*treehill*	*room*	*counter*	*kitchen*	*bonsai*
NeRF [9, 5]	0.536	0.529	0.415	0.551	0.546	0.353	0.394	0.335	0.398
mip-NeRF [2]	0.541	0.535	0.422	0.490	0.538	0.346	0.390	0.336	0.370
NeRF++ [16]	0.455	0.466	0.331	0.416	0.466	0.335	0.351	0.260	0.291
Instant NGP [11, 14]	0.398	0.441	0.255	0.339	0.420	0.242	0.255	0.170	0.198
mip-NeRF 360 [3, 10]	0.289	0.345	0.164	0.254	0.338	0.211	0.203	0.126	0.177
mip-NeRF 360 + iNGP	0.272	0.305	0.134	0.256	0.298	0.198	0.259	0.129	0.171
Our Model	0.208	0.273	0.118	0.193	0.242	0.196	0.185	0.116	0.173

Table 1: Per-scene performance on the dataset of "360" indoor and outdoor scenes from mip-NeRF 360 [3].

PSNR	*chair*	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*	*avg*
NeRF [9, 5]	34.17	25.08	30.39	36.82	33.31	30.03	34.78	29.30	31.74
mip-NeRF [2]	35.14	25.48	33.29	37.48	35.70	30.71	36.51	30.41	33.09
mip-NeRF 360 [3, 10], 256 hidden	35.03	25.73	32.61	37.44	36.10	30.31	36.22	29.98	32.93
mip-NeRF 360 [3, 10], 512 hidden	35.65	25.60	33.19	37.71	36.10	29.90	36.52	31.26	33.24
Our Model	34.84	25.84	33.90	37.14	34.84	31.66	35.15	31.38	33.10

SSIM	*chair*	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*	*avg*
NeRF [9, 5]	0.975	0.925	0.967	0.979	0.968	0.953	0.987	0.869	0.953
mip-NeRF [2]	0.981	0.932	0.980	0.982	0.978	0.959	0.991	0.882	0.961
mip-NeRF 360 [3, 10], 256 hidden	0.980	0.934	0.977	0.981	0.980	0.953	0.990	0.883	0.960
mip-NeRF 360 [3, 10], 512 hidden	0.983	0.931	0.979	0.982	0.980	0.949	0.991	0.893	0.961
Our Model	0.983	0.944	0.985	0.984	0.980	0.969	0.991	0.929	0.971

LPIPS	*chair*	*drums*	*ficus*	*hotdog*	*lego*	*materials*	*mic*	*ship*	*avg*
NeRF [9, 5]	0.026	0.071	0.032	0.030	0.031	0.047	0.012	0.150	0.050
mip-NeRF [2]	0.021	0.065	0.020	0.027	0.021	0.040	0.009	0.138	0.043
mip-NeRF 360 [3, 10], 256 hidden	0.021	0.064	0.024	0.027	0.018	0.047	0.011	0.135	0.043
mip-NeRF 360 [3, 10], 512 hidden	0.018	0.069	0.022	0.024	0.018	0.053	0.011	0.119	0.042
Our Model	0.017	0.050	0.015	0.020	0.019	0.032	0.007	0.091	0.031

Table 2: Per-scene and average performance on the Blender dataset from NeRF [9].