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

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

(a) Ground Truth (b) Our Model (c) mip-NeRF 360 [3] (d) iNGP [21] (e) mip-NeRF 360 + iNGP

Figure 1: (a) Test-set images from the mip-NeRF 360 dataset [3] with renderings from (b) our model and (c, d, e) three state-of-the-art baselines. Our model accurately recovers thin structures and finely detailed foliage, while the baselines either oversmooth or exhibit aliasing in the form of jaggies and missing scene content. PSNR values for each patch are inset.

Abstract

Neural Radiance Field training can be accelerated through the use of grid-based representations in NeRF's learned mapping from spatial coordinates to colors and volumetric density. However, these grid-based approaches lack an explicit understanding of scale and therefore often introduce aliasing, usually in the form of jaggies or missing scene content. Anti-aliasing has previously been addressed by mip-NeRF 360, which reasons about sub-volumes along a cone rather than points along a ray, but this approach is not natively compatible with current grid-based techniques. We show how ideas from rendering and signal processing can be used to construct a technique that combines mip-NeRF 360 and grid-based models such as Instant NGP to yield error rates that are 8% – 77% lower than either prior technique, and that trains $24\times$ *faster than mip-NeRF 360.*

In Neural Radiance Fields (NeRF), a neural network is trained to model a volumetric representation of a 3D scene such that novel views of that scene can be rendered via ray-tracing [19]. NeRF has proven to be an effective tool for tasks such as view synthesis [17], generative media [25], robotics [33], and computational photography [18].

The original NeRF model used a multilayer perceptron (MLP) to parameterize the mapping from spatial coordinates to colors and densities. Though compact and expressive, MLPs are slow to train, and recent work has accelerated training by replacing or augmenting MLPs with voxel-grid-like datastructures [7, 15, 27, 35]. One example is Instant NGP (iNGP), which uses a pyramid of coarse and fine grids (the finest of which are stored using a hash map) to construct learned features that are processed by a tiny MLP, enabling greatly accelerated training [21].

In addition to being slow, the original NeRF model was also aliased: NeRF reasons about individual points along a ray, which results in "jaggies" in rendered images and limits NeRFs ability to reason about scale. Mip-NeRF [2] resolved this issue by casting cones instead of rays, and by featurizing the entire volume within a conical frustum for

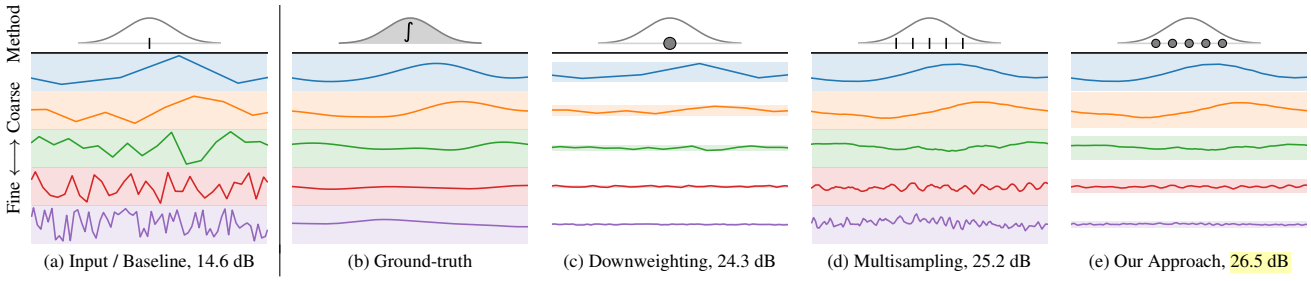


Figure 2: Here we show a toy 1-dimensional iNGP [21] with 1 feature per scale. Each subplot represents a different strategy for querying the iNGP at all coordinates along the x axis — imagine a Gaussian moving left to right, where each line is the iNGP feature for each coordinate, and where each color is a different scale in the iNGP. (a) The naive solution of querying the Gaussian's mean results in features with piecewise-linear kinks, where the high frequencies past the bandwidth of the Gaussian are large and inaccurate. (b) The true solution, obtained by convolving the iNGP features with a Gaussian — an intractable solution in practice — results in coarse features that are smooth but informative and fine features that are near 0. (c) We can suppress unreliable high frequencies by downweighing them based on the scale of the Gaussian (color bands behind each feature indicate the downweighting), but this results in unnaturally sharp discontinuities in coarse features. (d) Alternatively, supersampling produces reasonable coarse scales features but erratic fine-scale features. (e) We therefore multisample isotropic sub-Gaussians (5 shown here) and use each sub-Gaussian's scale to downweight frequencies.

use as input to the MLP. Mip-NeRF and its successor mip-NeRF 360 [3] showed that this approach enables highly accurate rendering on challenging real-world scenes.

Regrettably, the progress made on these two problems of fast training and anti-aliasing are, at first glance, incompatible with each other. This is because mip-NeRF's anti-aliasing strategy depends critically on the use of positional encoding [28, 30] to featurize a conical frustum into a discrete feature vector, but current grid-based approaches do not use positional encoding, and instead use learned features that are obtained by interpolating into a hierarchy of grids at a single 3D coordinate. Though anti-aliasing is a well-studied problem in rendering [8, 9, 26, 31], most approaches do not generalize naturally to inverse-rendering in grid-based NeRF models like iNGP.

In this work, we leverage ideas from multisampling, statistics, and signal processing to integrate iNGP's pyramid of grids into mip-NeRF 360's framework. We call our model "Zip-NeRF" due to its speed, its similarity with mip-NeRF, and its ability to fix zipper-like aliasing artifacts. On the mip-NeRF 360 benchmark [3], Zip-NeRF reduces error rates by as much as 19% and trains $24\times$ faster than the previous state-of-the-art. On our multiscale variant of that benchmark, which more thoroughly measures aliasing and scale, Zip-NeRF reduces error rates by as much as 77%.

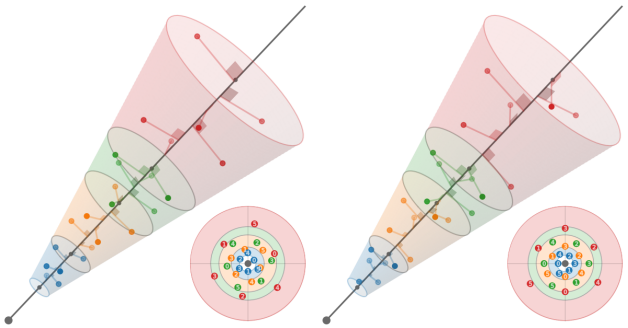
1. Preliminaries

Mip-NeRF 360 and Instant NGP (iNGP) are both NeRF-like [19]: A pixel is rendered by casting a 3D ray and featurizing locations at distances t along the ray, and those features are fed to a neural network whose outputs are alpha-composited to render a color. Training consists of repeatedly casting rays corresponding to pixels in training images and minimizing (via gradient descent) the difference between each pixel's rendered and observed colors.

Mip-NeRF 360 and iNGP differ significantly in how coordinates along a ray are parameterized. In mip-NeRF 360, a ray is subdivided into a set of intervals $[t_i, t_{i+1})$, each of which represents a conical frustum whose shape is approximated with a multivariate Gaussian, and the expected positional encoding with respect to that Gaussian is used as input to a large MLP [2]. In contrast, iNGP trilinearly interpolates into a hierarchy of differently-sized 3D grids to produce feature vectors for a small MLP [21]. Our model combines mip-NeRF 360's overall framework with iNGP's featurization approach, but naively combining these two approaches introduces two forms of aliasing:

1. Instant NGP's feature grid approach is incompatible with mip-NeRF 360's scale-aware integrated positional encoding technique, so the features produced by iNGP are aliased with respect to spatial coordinates and thus produce aliased renderings. In Section 2, we address this by introducing a multisampling-like solution for computing prefiltered iNGP features.

2. Using iNGP dramatically accelerates training, but this reveals a problem with mip-NeRF 360's online distillation approach that causes highly visible "z-aliasing" (aliasing along a ray), wherein scene content erratically disappears as the camera moves. In Section 3 we resolve this with a new loss function that prefilters along each ray when computing the loss function used to supervise online distillation.



(a) Training (b) Testing

Figure 3: Here we show a toy 3D ray with an exaggerated pixel width (viewed along the ray as an inset) divided into 4 frustums denoted by color. We multisample each frustum with a hexagonal pattern that matches the frustum's first and second moments. Each pattern is rotated around the ray and flipped along the ray (a) randomly when training and (b) deterministically when rendering.

2. Spatial Anti-Aliasing

Mip-NeRF uses features that approximate the integral over the positional encoding of the coordinates within a sub-volume, which is a conical frustum. This results in Fourier features whose amplitudes are small when the feature sinusoid's period is larger than the standard deviation of the Gaussian — the features express the spatial location of the sub-volume *only* at the wavelengths that are larger than the size of the sub-volume. Because this feature encodes both position and scale, the MLP that consumes it is able to learn a multi-scale representation of the 3D scene that renders anti-aliased images. Grid-based representations like iNGP do not natively allow for sub-volumes to be queried, and instead use trilinear interpolation at a single point to construct features for use in an MLP, which results in learned models that cannot reason about scale or aliasing.

We resolve this by turning each conical frustum into a set of isotropic Gaussians, using a combination of multisampling and feature-downweighting: The anisotropic sub-volume is first converted into a set of points that approximate its shape, and then each point is assumed to be an isotropic Gaussian with some scale. This isotropic assumption lets us approximate the true integral of the feature grid over a sub-volume by leveraging the fact that the values in the grid are zero-mean. By averaging these downweighted features, we obtain scale-aware prefiltered features from an iNGP grid. See Figure 2 for a visualization.

Anti-aliasing is well explored in the graphics literature. Mip mapping [31] (mip-NeRF's namesake) precomputes a datastructure that enables fast anti-aliasing, but it is unclear how this approach can be applied to iNGP's hash-based datastructures. Supersampling techniques [8] adopt a brute-force approach to anti-aliasing and use a large number of samples; we will demonstrate that this is less effective and more expensive than our approach. Multisampling techniques [11] construct a small set of samples, and then pool information from those multisamples into an aggregate representation that is provided to an expensive rendering process — a strategy that resembles our approach.

Multisampling Following mip-NeRF [2], we assume each pixel corresponds to a cone with radius $\dot{r}t$, where t is distance along the ray. Given an interval along the ray $[t_0, t_1)$, we would like to construct a set of multisamples that approximate the shape of that conical frustum. We use a 6-point hexagonal pattern whose angles θ_j are:

$$\boldsymbol{\theta} = [\,0, {}^{2\pi}/_{3}, {}^{4\pi}/_{3}, {}^{3\pi}/_{3}, {}^{5\pi}/_{3}, {}^{\pi}/_{3}\,], \tag{1}$$

which are linearly-spaced angles around one rotation of a circle, permuted to give a pair of triangles that are shifted by 60 degrees. The distances along the ray t_j are:

$$t_j = t_0 + \frac{t_\delta\left(t_1^2 + 2t_\mu^2 + \frac{3}{\sqrt{7}}\left(\frac{2j}{5} - 1\right)\sqrt{\left(t_\delta^2 - t_\mu^2\right)^2 + 4t_\mu^4}\right)}{t_\delta^2 + 3t_\mu^2}$$
$$\text{where } t_\mu = (t_0 + t_1)/2\,, \quad t_\delta = (t_1 - t_0)/2 \tag{2}$$

which are linearly-spaced values in $[t_0, t_1)$, shifted and scaled to concentrate mass near the far base of the frustum. Our multisample coordinates relative to the ray are:

$$\left\{ \begin{bmatrix} \dot{r}t_j \cos(\theta_j)/\sqrt{2} \\ \dot{r}t_j \sin(\theta_j)/\sqrt{2} \\ t_j \end{bmatrix} \middle| \, j = 0, 1, \ldots, 5 \right\}. \tag{3}$$

These 3D coordinates are rotated into world coordinates by multiplying them by an orthonormal basis whose third vector is the ray direction (and whose first two vectors are an arbitrary frame that is perpendicular to the ray) and then shifted by the ray origin. By construction, the sample means and variances (along the ray and perpendicular to the ray) of these multisamples exactly match those of the conical frustum, analogously to mip-NeRF Gaussians [2] (see the supplement for details). During training, we randomly rotate and flip each pattern, and during rendering we deterministically flip and rotate every other pattern by 30 degrees. See Figure 3 for visualizations of both strategies.

We use these 6 multisamples $\{\mathbf{x}_j\}$ as the means of isotropic Gaussians each with a standard deviation of σ_j. We set σ_j to $\dot{r}t_j/\sqrt{2}$ scaled by a hyperparameter, which is 0.5 in all experiments. Because iNGP grids require input coordinates to lie in a bounded domain, we apply the contraction function from mip-NeRF 360 [3]. Since these Gaussians are isotropic, we can compute the scale factor induced by this contraction using an efficient alternative to the Kalman filter approach used by mip-NeRF 360; see the supplement for details.

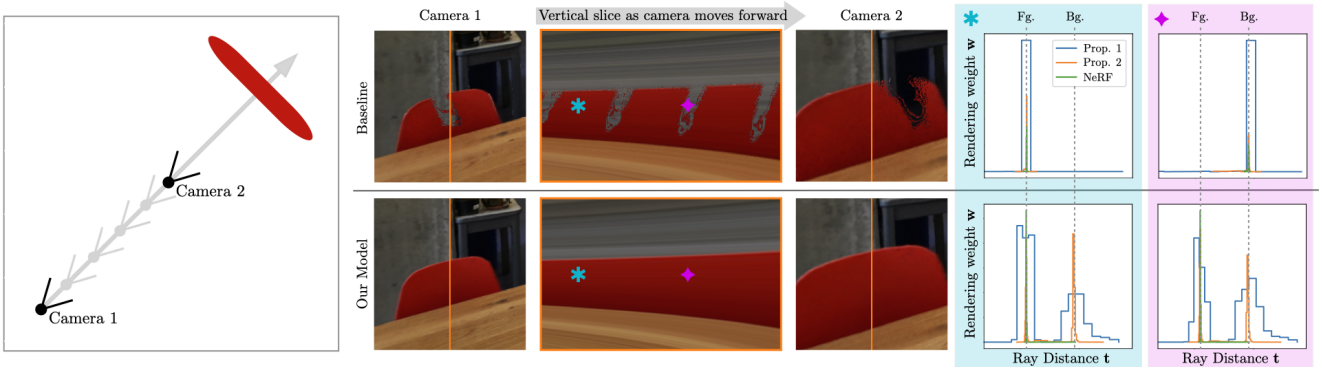


Figure 4: Here we visualize the problem of z-aliasing. Left: we have a scene where 2 training cameras face a narrow red chair in front of a gray wall. Middle: As we sweep a test camera between those training cameras, we see that the baseline algorithm (top) "misses" or "hits" the chair depending on its distance and therefore introduces tearing artifacts, while our model (bottom) consistently "hits" the chair to produce artifact-free renderings. Left: This is because the baseline (top) has learned non-smooth proposal distributions due to aliasing in its supervision, while our model (bottom) correctly predicts proposal distributions that capture both the foreground and the background at all depths due to our anti-aliased loss function.

Downweighting Though multisampling is an effective tool for reducing aliasing, using naive trilinear interpolation for each multisample still results in aliasing at high frequencies, as can be seen in Figure 2(d). To anti-alias interpolation *for each individual multisample*, we re-weight the features at each scale in a way that is inversely-proportional to how much each multisample's isotropic Gaussian fits within each grid cell: if the Gaussian is much larger than the cell being interpolated into, the interpolated feature is likely unreliable and should be downweighted. Mip-NeRF's IPE features have a similar interpretation.

In iNGP, interpolation into each $\{V_\ell\}$ at coordinate $\mathbf{x}$ is done by scaling $\mathbf{x}$ by the grid's linear size n_ℓ and performing trilinear interpolation into V_ℓ to get a c_ℓ-length vector. We instead interpolate a set of multisampled isotropic Gaussians with means $\{\mathbf{x}_j\}$ and standard deviations $\{\sigma_j\}$. By reasoning about Gaussian CDFs we can compute the fraction of each Gaussian's PDF that is inside the $[-1/2n, 1/2n]^3$ cube in V_ℓ being interpolated into as a scale-dependent downweighting factor $\omega_{j,\ell} = \text{erf}(1/\sqrt{8\sigma_j^2 n_\ell^2})$, where $\omega_{j,\ell} \in [0, 1]$. As detailed in Section 4, we impose weight decay on $\{V_\ell\}$, which encourages the values in V_ℓ to be normally distributed and zero-mean. This zero-mean assumption lets us approximate the expected grid feature with respect to each multisample's Gaussian as $\omega_j \cdot \mathbf{f}_{j,\ell} + (1-\omega_j) \cdot \mathbf{0} = \omega_j \cdot \mathbf{f}_{j,\ell}$. With this, we can approximate the expected feature corresponding to the conical frustum being featurized by taking a weighted mean of each multisample's interpolated features:

$$\mathbf{f}_\ell = \text{mean}_j(\omega_{j,\ell} \cdot \text{trilerp}(n_\ell \cdot \mathbf{x}_j; V_\ell)) \,. \tag{4}$$

This set of features $\{\mathbf{f}_\ell\}$ is concatenated and provided as input to an MLP, as in iNGP. We also concatenate featurized versions of $\{\omega_{j,\ell}\}$, see the supplement for details.

3. Z-Aliasing and Proposal Supervision

Though the previously-detailed multisampling and downweighting approach is an effective way to reduce spatial aliasing, there is an additional source of aliasing *along the ray* that we must consider, which we will refer to as z-aliasing. This z-aliasing is due to mip-NeRF 360's use of a proposal MLP that learns to produce upper bounds on scene geometry: during training and rendering, intervals along a ray are repeatedly evaluated by this proposal MLP to generate histograms that are resampled by the next round of sampling, and only the final set of samples is rendered by the NeRF MLP. Mip-NeRF 360 showed that this approach significantly improves speed and rendering quality compared to prior strategies of learning one [2] or multiple [19] NeRFs that are all supervised using image reconstruction. We observe that the proposal MLP in mip-NeRF 360 tends to learn a non-smooth mapping from input coordinates to output volumetric densities. This results in artifacts in which a ray "skips" scene content, which can be seen in Figure 4. Though this artifact is subtle in mip-NeRF 360, if we use an iNGP backend for our proposal network instead of an MLP (thereby increasing our model's ability to optimize rapidly) it becomes common and visually salient, especially when the camera translates along its z-axis.

The root cause of z-aliasing in mip-NeRF 360 is the "interlevel loss" used to supervise the proposal network, which assigns an equivalent loss to NeRF and proposal histogram bins whether their overlap is partial or complete — proposal histogram bins are only penalized when they completely fail to overlap. To resolve this issue, we present an alternative loss that, unlike mip-NeRF 360's interlevel loss, is continuous and smooth with respect to distance along the ray. See

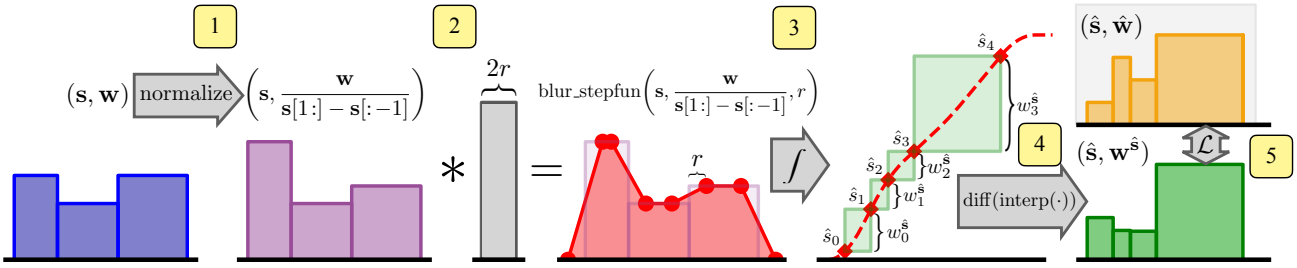
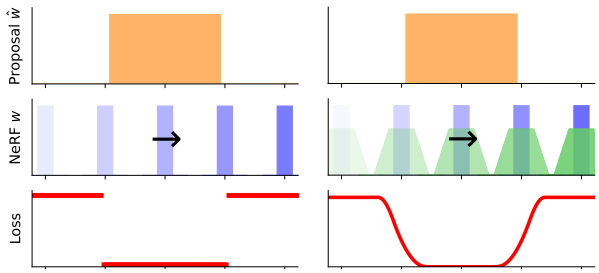


Figure 5: Computing our anti-aliased loss requires that we smooth and resample a NeRF histogram $(\mathbf{s}, \mathbf{w})$ into the same set of endpoints as a proposal histogram $(\hat{\mathbf{s}}, \hat{\mathbf{w}})$, which we outline here. (1) We divide $\mathbf{w}$ by the size of each interval in $\mathbf{s}$ to yield a piecewise constant PDF that integrates to ≤ 1. (2) We convolve that PDF with a rectangular pulse to obtain a piecewise linear PDF. (3) This PDF is integrated to produce a piecewise-quadratic CDF that is queried via piecewise quadratic interpolation at each location in $\hat{\mathbf{s}}$. (4) By taking the difference between adjacent interpolated values we obtain $\mathbf{w}^{\hat{\mathbf{s}}}$, which are the NeRF histogram weights $\mathbf{w}$ resampled into the endpoints of the proposal histogram $\hat{\mathbf{s}}$. (5) After resampling, we evaluate our loss $\mathcal{L}_{\text{prop}}$ as an element-wise function of $\mathbf{w}^{\hat{\mathbf{s}}}$ and $\hat{\mathbf{w}}$, as they share a common coordinate space.

Figure 6 for a comparison of both loss functions.

Blurring a Step Function To design a loss function that is smooth with respect to distance along the ray, we must first construct a technique for turning a piecewise constant step function into a continuous piecewise-linear function. Smoothing a discrete 1D signal is trivial, and requires just discrete convolution with a box filter (or a Gaussian filter, *etc*.). But this problem becomes difficult when dealing with piecewise constant functions whose endpoints are *continuous*, as the endpoints of each interval in the step function may be at any location. Rasterizing the step function and performing convolution is not a viable solution, as this fails in the common case of extremely narrow histogram bins.

A new algorithm is therefore required. Consider a step function $(\mathbf{x}, \mathbf{y})$ where (x_i, x_{i+1}) are the endpoints of interval i and y_i is the value of the step function inside interval i. We want to convolve this step function with a rectangular pulse with radius r that integrates to 1: $[|\mathbf{x}| < r]/(2r)$ where $[\,]$ are Iverson brackets. Observe that the convolution of a single interval i of a step function with this rectangular pulse is a piecewise linear trapezoid with knots at $(x_i - r, 0), (x_i + r, y_i), (x_{i+1} - r, y_i), (x_{i+1} + r, 0)$, and that a piecewise linear spline is the double integral of scaled delta functions located at each spline knot [12]. With this, and with the fact that summation commutes with integration, we can efficiently convolve a step function with a rectangular pulse as follows: We turn each endpoint x_i of the step function into two signed delta functions located at $x_i - r$ and $x_i + r$ with values that are proportional to the change between adjacent y values, we interleave (via sorting) those delta functions, and we then integrate those sorted delta functions twice (see Algorithm 1 of the supplement for pseudocode). With this, we can construct our anti-aliased loss function.



(a) mip-NeRF 360 (b) Our Approach

Figure 6: Here we visualize proposal supervision for a toy setting where a narrow NeRF histogram (blue) translates along a ray relative to a coarse proposal histogram (orange). (a) The loss used by mip-NeRF 360 is piecewise constant, but (b) our loss is smooth because we blur NeRF histograms into piecewise linear splines (green). Our prefiltered loss lets us learn anti-aliased proposal distributions.

Anti-Aliased Interlevel Loss The proposal supervision approach in mip-NeRF 360, which we inherit, requires a loss function that takes as input a step function produced by the NeRF $(\mathbf{s}, \mathbf{w})$ and a similar step function produced by the proposal model $(\hat{\mathbf{s}}, \hat{\mathbf{w}})$. Both of these step functions are histograms, where $\mathbf{s}$ and $\hat{\mathbf{s}}$ are vectors of endpoint locations while $\mathbf{w}$ and $\hat{\mathbf{w}}$ are vectors of weights that sum to ≤ 1, where w_i denote how visible scene content is interval i of the step function. Each s_i is some normalized function of true metric distance t_i, according to some normalization function $g(\cdot)$, which we discuss later. Note that $\mathbf{s}$ and $\hat{\mathbf{s}}$ are not the same — the endpoints of each histogram are distinct.

To train a proposal network to bound the scene geometry predicted by the NeRF without introducing aliasing, we require a loss function that can measure the distance

between $(\mathbf{s}, \mathbf{w})$ and $(\hat{\mathbf{s}}, \hat{\mathbf{w}})$ that is smooth with respect to translation along the ray, despite the challenge that the endpoints of both step functions are different. To do this, we will blur the NeRF histogram $(\mathbf{s}, \mathbf{w})$ using our previously-constructed algorithm and then resample that blurred distribution into the set of intervals of the proposal histogram $\hat{\mathbf{s}}$ to produce a new set of histogram weights $\mathbf{w}^{\hat{\mathbf{s}}}$. This procedure is described in Figure 5. After resampling our blurred NeRF weights into the space of our proposal histogram, our loss function is an element-wise function of $\mathbf{w}^{\hat{\mathbf{s}}}$ and $\hat{\mathbf{w}}$:

$$\mathcal{L}_{\text{prop}}(\mathbf{s}, \mathbf{w}, \hat{\mathbf{s}}, \hat{\mathbf{w}}) = \sum_i \frac{1}{\hat{w}_i} \max(0, \not{\nabla}(w_i^{\hat{\mathbf{s}}}) - \hat{w}_i)^2 \,. \quad (5)$$

Though this loss resembles mip-NeRF 360's (a half-quadratic chi-squared loss with a stop-gradient), the blurring and resampling used to generate $\mathbf{w}^{\hat{\mathbf{s}}}$ prevents aliasing.

Normalizing Metric Distance As in mip-NeRF 360, we parameterize metric distance along a ray $t \in [t_{near}, t_{far}]$ in terms of normalized distance $s \in [0, 1]$ (where t_{near} and t_{far} are manually-defined near and far plane distances). Rendering uses metric distance t, but resampling and proposal supervision use normalized distance s, with some function $g(\cdot)$ defining a bijection between the two. The interlevel loss used in mip-NeRF 360 is invariant to monotonic transformations of distance, so it is unaffected by the choice of $g(\cdot)$. However, the prefiltering in our anti-aliased loss removes this invariance, and using mip-NeRF 360's $g(\cdot)$ in our model results in catastrophic failure, so we must construct a new normalization. To do this, we construct a novel power transformation [5, 29]:

$$\mathcal{P}(x, \lambda) = \frac{|\lambda - 1|}{\lambda} \left(\left(\frac{x}{|\lambda - 1|} + 1 \right)^{\lambda} - 1 \right). \quad (6)$$

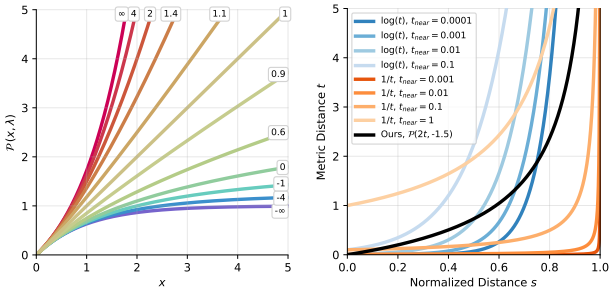


Figure 7: Many NeRF approaches require a function for converting metric distance $t \in [0, \infty)$ into normalized distance $s \in [0, 1]$. Left: Our power transformation $\mathcal{P}(x, \lambda)$ lets us interpolate between common curves such as linear, logarithmic, and inverse by modifying λ, while retaining a linear-like shape near the origin. Right: This lets us construct a curve that transitions from linear to inverse/inverse-square, and supports scene content close to the camera.

The slope of this function at the origin is 1, so normalized distance near the ray origin is proportional to metric distance (obviating the need to tune a non-zero near plane distance t_{near}), but far from the origin metric distance becomes curved to resemble log-distance [22] ($\lambda = 0$) or inverse-distance [3] ($\lambda = -1$). This lets us smoothly interpolate between different normalizations, instead of swapping in different discrete functions. See Figure 7 for a visualization of $\mathcal{P}(x, \lambda)$ and a comparison of prior normalization approaches to ours, which is $g(x) = \mathcal{P}(2x, -1.5)$ — a curve that is roughly linear when $s \in [0, 1/2)$ but is between inverse and inverse-square when $s \in [1/2, 1]$.

4. Results

Our model is implemented in JAX [6] and based on the mip-NeRF 360 codebase [20], with a reimplementation of iNGP's pyramid of voxel grids and hashes in place of the large MLP used by mip-NeRF 360. Our overall model architecture is identical to mip-NeRF 360 except for the anti-aliasing adjustments introduced in Sections 2 and 3, as well as some additional modifications that we describe here.

Like mip-NeRF 360, we use two rounds of proposal sampling with 64 samples each, and then 32 samples in the final NeRF sampling round. Our anti-aliased interlevel loss is imposed on both rounds of proposal sampling, with a rectangular pulse width of $r = 0.03$ in the first round and $r = 0.003$ in the second round, and with a loss multiplier of 0.01. We use separate proposal iNGPs and MLPs for each round of proposal sampling, and our NeRF MLP uses a much larger view-dependent branch than is used by iNGP. See the supplement for details.

One small but important modification we make to iNGP is imposing a normalized weight decay on the feature codes stored in its pyramid of grids and hashes: $\sum_\ell \text{mean}(V_\ell^2)$. By penalizing *the sum of the mean* of squared grid/hash values at each pyramid level V_ℓ we induce very different behavior than the naive solution of penalizing the sum of all values, as coarse scales are penalized by orders of magnitude more than fine scales. This simple trick is extremely effective — it improves performance greatly compared to no weight decay, and significantly outperforms naive weight decay. We use a loss multiplier of 0.1 on this normalized weight decay in all experiments.

360 Dataset Since our model is an extension of mip-NeRF 360, we evaluate on the "360" benchmark presented in that paper using the same evaluation procedure [3]. We evaluate NeRF [19], mip-NeRF [2], NeRF++ [36], mip-NeRF 360, iNGP [21] (using carefully tuned hyperparameters taken from recent work [32]), and a baseline in which we naively combine mip-NeRF 360 and iNGP without any of this paper's contributions. Average error metrics across

Scale Factor:	1×			2×			4×			8×			
Error Metric:	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	Time (hrs)
Instant NGP [21, 32]	24.36	0.642	0.366	25.23	0.712	0.251	26.84	0.809	0.142	28.42	0.877	0.092	0.15
mip-NeRF 360 [3, 20]	27.51	0.779	0.254	29.19	0.864	0.136	30.45	0.912	0.077	30.86	0.931	0.058	21.86
mip-NeRF 360 + iNGP	26.46	0.773	0.253	27.92	0.855	0.141	27.67	0.866	0.116	25.58	0.804	0.160	0.31
Our Model	28.25	0.822	0.198	30.00	0.892	0.099	31.57	0.933	0.056	32.52	0.954	0.037	0.90
A) Naive Sampling	27.93	0.797	0.233	29.70	0.880	0.114	29.24	0.887	0.094	26.53	0.820	0.144	0.53
B) Naive Supersampling (6×)	27.48	0.803	0.224	29.03	0.881	0.109	28.42	0.881	0.097	25.97	0.810	0.151	2.54
C) Jittered	27.91	0.797	0.233	29.60	0.879	0.116	29.45	0.893	0.090	27.58	0.855	0.120	0.55
D) Jittered Supersampling (6×)	27.50	0.810	0.212	28.99	0.884	0.105	28.91	0.896	0.086	27.65	0.870	0.109	3.04
E) No Multisampling	28.15	0.817	0.208	29.87	0.886	0.105	31.33	0.927	0.061	32.12	0.948	0.043	0.54
F) No Downweighting	28.22	0.818	0.205	29.94	0.889	0.102	31.25	0.928	0.060	31.67	0.944	0.046	0.88
G) No Appended Scale ω	28.23	0.820	0.200	29.98	0.890	0.101	31.48	0.931	0.057	32.19	0.951	0.041	0.89
H) Random Multisampling	28.09	0.816	0.207	29.75	0.886	0.106	31.18	0.928	0.061	32.03	0.950	0.042	0.95
I) Unscented Multisampling	28.27	0.822	0.198	30.03	0.891	0.100	31.57	0.933	0.056	32.46	0.954	0.038	1.11
J) No New Interlevel Loss	28.12	0.824	0.196	29.82	0.892	0.098	31.31	0.932	0.056	32.23	0.953	0.039	0.86
K) No Weight Decay	27.34	0.814	0.203	28.91	0.881	0.109	30.29	0.921	0.067	31.23	0.941	0.050	0.90
L) Un-Normalized Weight Decay	27.99	0.821	0.196	29.65	0.889	0.100	31.10	0.930	0.058	32.09	0.951	0.040	0.91
M) Small View-Dependent MLP	27.41	0.811	0.207	28.98	0.882	0.109	30.32	0.924	0.065	31.11	0.944	0.047	0.63

Table 1: Performance on our multiscale version of the 360 dataset [3], where we train and evaluate on multiscale images [2]. Red, orange, and yellow highlights indicate the 1st, 2nd, and 3rd-best performing technique for each metric. Our model significantly outperforms our two baselines — especially the iNGP-based one, and especially at coarse scales, where our errors are 55%–77% reduced. Rows A–M are ablations of our model, see the text for details.

all scenes are shown in Table 2 (see the supplement for per-scene metrics) and renderings of these four approaches are shown in Figure 1. Our model, mip-NeRF 360, and our "mip-NeRF 360 + iNGP" baseline were all trained on 8 NVIDIA Tesla V100-SXM2-16GB GPUs. The other baselines were trained on different accelerators (TPUs were used for NeRF, mip-NeRF, and NeRF++, and a single NVIDIA 3090 was used for iNGP) so to enable comparison their runtimes have been rescaled to approximate performance on our hardware. The render time of our model (which is not a focus of this work) is 0.9s, while mip-NeRF 360 takes 7.4s and our mip-NeRF 360 + iNGP baseline takes 0.2s.

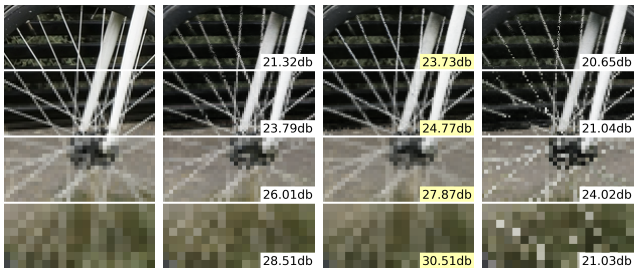
Our model significantly outperforms mip-NeRF 360 (the previous state-of-the-art of this task) on this benchmark: we observe 11%, 17%, and 19% reductions in RMSE, DSSIM, and LPIPS respectively. Additionally, our model trains 24× faster than mip-NeRF 360. Compared to iNGP, our error reduction is even more significant: 28%, 42%, and 37% reductions in RMSE, DSSIM, and LPIPS, though our model is ∼6× slower to train than iNGP. Our combined "mip-NeRF 360 + iNGP" baseline trains significantly faster than mip-NeRF 360 (and is nearly 3× faster to train than our model) and produces error rates comparable to mip-NeRF 360's and better than iNGP's. Our model's improvement in image quality over these baselines is clear upon visual inspection, as shown in Figure 1 and the supplemental video.

Multiscale 360 Dataset Though the 360 dataset contains challenging scene content, it does not measure rendering quality as a function of *scale* — because this dataset was captured by orbiting a camera around a central object at a roughly constant distance, learned models need not generalize well across different image resolutions or different distances from the central object. We therefore use a more challenging evaluation procedure, similar to the multiscale Blender dataset used by mip-NeRF [2]: we turn each image into a set of four images that have been bicubically downsampled by scale factors $[1, 2, 4, 8]$ where the downsampled images act as a proxy for additional training/test views in which the camera has been zoomed out from the center of the scene. During training we multiply the data term by the scale factor of each ray, and at test time we evaluate each scale separately. This significantly increases the difficulty of reconstruction by requiring the learned model to generalize across scales, and it causes aliasing artifacts to be highly salient, especially at coarse scales.

In Table 1 we evaluate our model against iNGP, mip-NeRF 360, our mip-NeRF 360 + iNGP baseline, and many ablations. Though mip-NeRF 360 performs reasonably (as it can reason about scale) our model still reduces RMSE by 8% at the finest scale and 17% at the coarsest scale, while being 24× faster. The mip-NeRF 360 + iNGP baseline, which has no mechanism for anti-aliasing or reasoning about scale, performs poorly: our RMSEs are 19% lower at the finest scale and 55% lower at the coarsest scale, and

	PSNR ↑	SSIM ↑	LPIPS ↓	Time (hrs)
NeRF [19, 10]	23.85	0.605	0.451	12.65
mip-NeRF [2]	24.04	0.616	0.441	9.64
NeRF++ [36]	25.11	0.676	0.375	28.73
Instant NGP [21, 32]	25.68	0.705	0.302	0.15
mip-NeRF 360 [3, 20]	27.57	0.793	0.234	21.69
mip-NeRF 360 + iNGP	26.43	0.786	0.225	0.30
Our Model	28.54	0.828	0.189	0.89

Table 2: Performance on the 360 dataset [3]. Runtimes shown in gray are approximate, see text for details.



(a) Ground Truth (b) mip-NeRF 360 (c) Our Model (d) mip360 + iNGP

Figure 8: A test-set view from our multiscale benchmark at (a) the four different scales that we train and test on. (b) Mip-NeRF 360 behaves reasonably across scales but is outperformed by (c) our model, especially on thin structures. (d) Our iNGP-based baseline produces extremely aliased renderings, especially at coarse scales. PSNRs are inset.

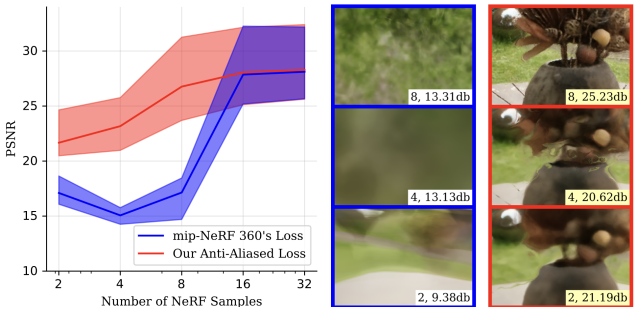


Figure 9: Our anti-aliased interlevel loss function (red) lets performance degrades gracefully as the number of NeRF samples is reduced from 32 to 2. Mip-NeRF 360's loss (blue) results in catastrophic failure if fewer than 16 samples are used. Here we plot the mean and IQR of test-set PSNR. NeRF sample counts and PSNRs are inset.

both DSSIM and LPIPS at the coarsest scale are 77% lower. This improvement can be seen in Figure 8. Our mip-NeRF 360 + iNGP baseline generally outperforms iNGP (except at the coarsest scale), as we might expect from Table 2.

In Table 1 we also include an ablation study of our model. (A) Using "naive" sampling with our multisampling and downweighting disabled (which corresponds to the approach used in NeRF and iNGP) significantly degrades quality, and (B) supersampling [8] by a $6\times$ factor to match our multisample budget does not improve accuracy. (C, D) Randomly jittering these naive samples within the cone being cast improves performance at coarse scales, but only slightly. Individually disabling (E) multisampling and (F) downweighting lets us measure the impact of each component; they both contribute evenly. (G) Not using ω as a feature hurts performance slightly. Alternative multisampling approaches like (H) sampling 6 random points from each mip-NeRF Gaussian or (G) using 7 unscented transform control points from each Gaussian [14] as multisamples are competitive alternatives to our hexagonal pattern, but have slightly worse speeds and/or accuracies. (J) Disabling our anti-aliased interlevel loss has little effect on our single-image metrics, but causes z-aliasing artifacts in our video results. (K) Disabling our normalized weight decay reduces accuracy significantly, and (L) using non-normalized weight decay performs poorly. (M) Using iNGP's small view-dependent MLP decreases accuracy.

Sample Efficiency Ablating our anti-aliased interlevel loss in our multiscale benchmark does not degrade quality significantly because mip-NeRF 360 (and our model, for the sake of fair comparison) uses a large number of samples[1] along each ray, which reduces z-aliasing. To reveal mip-NeRF 360's vulnerability to z-aliasing, in Figure 9 we plot test-set PSNR for our multiscale benchmark as we reduce the number of samples along the ray (both for the NeRF and the proposal network) by factors of 2, while also doubling r and the number of training iterations. Though our anti-aliased loss only slightly outperforms mip-NeRF 360's loss when the number of samples is large, as the sample count decreases the baseline loss fails catastrophically while our loss degrades gracefully.

That said, the most egregious z-aliasing artifact of missing scene content (shown in Figure 4) is hard to measure in small benchmarks of still images, as scene content only disappears at certain distances that the test set may not probe. See the supplemental video for demonstrations of z-aliasing, and for renderings from our model where that aliasing has been fixed.

5. Conclusion

We have presented Zip-NeRF, a model that integrates the progress made in the formerly divergent areas of scale-aware anti-aliased NeRFs and fast grid-based NeRF training. By leveraging ideas about multisampling and prefiltering, our model is able to achieve error rates that are 8% – 77% lower than prior techniques, while also training $24\times$ faster than mip-NeRF 360 (the previous state-of-the-art on our benchmarks). We hope that the tools and analysis presented here concerning aliasing (both the spatial aliasing of NeRF's learned mapping from spatial coordinate to color and density, and z-aliasing of the loss function used during online distillation along each ray) enable further progress towards improving the quality, speed, and sample-efficiency of NeRF-like inverse rendering techniques.

[1] Note that here we use "samples" to refer to sampled sub-frusta along the ray, not the sampled multisamples *within* each sub-frustum.

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] George E. P. Box and David R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1964.

[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew J. Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. *ECCV*, 2022.

[8] Robert L Cook. Stochastic sampling in computer graphics. *TOG*, 1986.

[9] Franklin C. Crow. Summed-area tables for texture mapping. *SIGGRAPH*, 1984.

[10] 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`.

[11] Ned Greene and Paul S. Heckbert. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications*, 1986.

[12] Paul S Heckbert. Filtering by repeated integration. *SIGGRAPH*, 1986.

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

[14] Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 2004.

[15] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. ReLU fields: The little non-linearity that could. *SIGGRAPH*, 2022.

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

[17] 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.

[18] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, and Jonathan T. Barron. NeRF in the dark: High dynamic range view synthesis from noisy raw images. *CVPR*, 2022.

[19] 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.

[20] 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.

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

[22] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum*, 2021.

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

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

[25] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D Diffusion. *ICLR*, 2023.

[26] Peter Shirley. Discrepancy as a quality measure for sample distributions. *Eurographics*, 1991.

[27] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. *CVPR*, 2022.

[28] 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. *NeurIPS*, 2020.

[29] John W. Tukey. *Exploratory Data Analysis*. 1977.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[31] Lance Williams. Pyramidal parametrics. *Computer Graphics*, 1983.

[32] 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.

[33] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Tsung-Yi Lin, Alberto Rodriguez, and Phillip Isola. NeRF-Supervision: Learning dense object descriptors from neural radiance fields. *ICRA*, 2022.

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

[35] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *CVPR*, 2022.

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

A. 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 [17], 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 [13, 23, 24]. 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.

B. 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:

$$\left\{ \begin{bmatrix} \cos(\theta_j)/\sqrt{3} \\ \sin(\theta_j)/\sqrt{3} \\ \frac{j-(n-1)/2}{\sqrt{n(n^2-1)/12}} \end{bmatrix} \middle| \, j = 0, 1, \ldots, n \right\}. \tag{7}$$

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

$$\boldsymbol{\theta}_1 = [\, 0, {}^{2\pi}/_3, {}^{4\pi}/_3, {}^{3\pi}/_3, {}^{5\pi}/_3, {}^{\pi}/_3 \,], \tag{8}$$

$$\boldsymbol{\theta}_2 = [\, 0, {}^{3\pi}/_3, {}^{2\pi}/_3, {}^{5\pi}/_3, {}^{4\pi}/_3, {}^{\pi}/_3 \,]. \tag{9}$$

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).

C. 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:

$$(2 \cdot \text{mean}_j(\omega_{j,\ell}) - 1)\sqrt{V_{init}^2 + \not\nabla(\text{mean}(V_\ell^2))}\,, \quad (10)$$

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 $\text{erf}(x)$:

$$\text{erf}(x) \approx \text{sign}(x)\sqrt{1 - \exp(-(4/\pi)x^2)}\,. \quad (11)$$

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

D. Spatial Contraction

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

$$\mathcal{C}(\mathbf{x}) = \begin{cases} \mathbf{x} & \|\mathbf{x}\| \le 1 \\ \left(2 - \frac{1}{\|\mathbf{x}\|}\right)\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right) & \|\mathbf{x}\| > 1\,. \end{cases} \quad (12)$$

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):

$$\mathcal{C}(\sigma_j) = \sigma_j |\det(\mathbf{J}_\mathcal{C}(\mathbf{x}_j))|^{1/d}\,. \quad (13)$$

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:

$$|\det(\mathbf{J}_\mathcal{C}(\mathbf{x}_j))|^{1/3} = \left(\frac{\sqrt[3]{2\max(1, \|\mathbf{x}_j\|) - 1}}{\max(1, \|\mathbf{x}_j\|)}\right)^2 \quad (14)$$

E. 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()`.

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

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

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

$$\mathcal{P}(x, \lambda) = \begin{cases} x & \lambda = 1 \\ \log(1 + x) & \lambda = 0 \\ e^x - 1 & \lambda = \infty \\ 1 - e^{-x} & \lambda = -\infty \\ \frac{|\lambda - 1|}{\lambda}\left(\left(\frac{x}{|\lambda - 1|} + 1\right)^\lambda - 1\right) & \text{otherwise} \end{cases} \quad (15)$$

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 [34] 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 10 for a visualization of our curve and the impact it has on distortion loss.

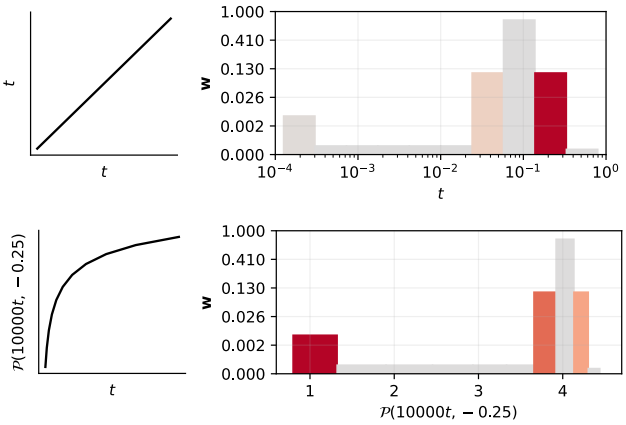


Figure 10: The behavior of mip-NeRF 360's distortion loss can be significantly modified by applying a curve to metric distance, which we do with our power transformation. Top: using a linear curve (*i.e.*, using metric distance t itself) results in a distortion loss that heavily penalizes distant scene 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.

G. 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 [16] 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 multi-

plier 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.

H. Results

An per-scene version of our single-scale results on the mip-NeRF 360 dataset can be found in Table 3.

We also include per-scene and average results for the Blender dataset [19] in Table 4. 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.

PSNR	Outdoor					Indoor			
	bicycle	*flowers*	*garden*	*stump*	*treehill*	*room*	*counter*	*kitchen*	*bonsai*
NeRF [19, 10]	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++ [36]	22.64	20.31	24.32	24.34	22.20	28.87	26.38	27.80	29.15
Instant NGP [21, 32]	22.79	19.19	25.26	24.80	22.46	30.31	26.21	29.00	31.08
mip-NeRF 360 [3, 20]	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 [19, 10]	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++ [36]	0.526	0.453	0.635	0.594	0.530	0.852	0.802	0.816	0.876
Instant NGP [21, 32]	0.540	0.378	0.709	0.654	0.547	0.893	0.845	0.857	0.924
mip-NeRF 360 [3, 20]	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 [19, 10]	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++ [36]	0.455	0.466	0.331	0.416	0.466	0.335	0.351	0.260	0.291
Instant NGP [21, 32]	0.398	0.441	0.255	0.339	0.420	0.242	0.255	0.170	0.198
mip-NeRF 360 [3, 20]	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 3: 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 [19, 10]	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, 20], 256 hidden	35.03	25.73	32.61	37.44	36.10	30.31	36.22	29.98	32.93
mip-NeRF 360 [3, 20], 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 [19, 10]	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, 20], 256 hidden	0.980	0.934	0.977	0.981	0.980	0.953	0.990	0.883	0.960
mip-NeRF 360 [3, 20], 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 [19, 10]	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, 20], 256 hidden	0.021	0.064	0.024	0.027	0.018	0.047	0.011	0.135	0.043
mip-NeRF 360 [3, 20], 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 4: Per-scene and average performance on the Blender dataset from NeRF [19].