Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields

Jonathan T. Barron[1] Ben Mildenhall[1] Dor Verbin[1,2]
Pratul P. Srinivasan[1] Peter Hedman[1]
[1]Google [2]Harvard University

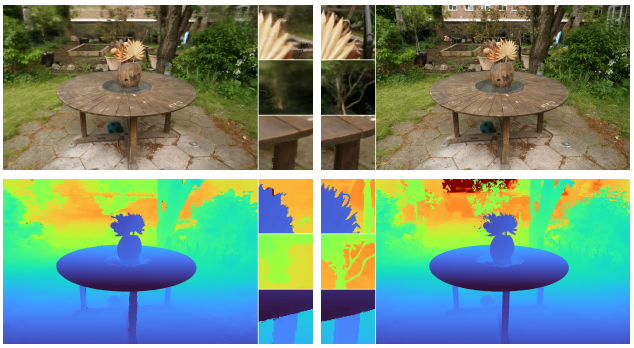
Abstract

Though neural radiance fields (NeRF) have demonstrated impressive view synthesis results on objects and small bounded regions of space, they struggle on "unbounded" scenes, where the camera may point in any direction and content may exist at any distance. In this setting, existing NeRF-like models often produce blurry or low-resolution renderings (due to the unbalanced detail and scale of nearby and distant objects), are slow to train, and may exhibit artifacts due to the inherent ambiguity of the task of reconstructing a large scene from a small set of images. We present an extension of mip-NeRF (a NeRF variant that addresses sampling and aliasing) that uses a non-linear scene parameterization, online distillation, and a novel distortion-based regularizer to overcome the challenges presented by unbounded scenes. Our model, which we dub "mip-NeRF 360" as we target scenes in which the camera rotates 360 degrees around a point, reduces mean-squared error by 57% compared to mip-NeRF, and is able to produce realistic synthesized views and detailed depth maps for highly intricate, unbounded real-world scenes.

Neural Radiance Fields (NeRF) synthesize highly realistic renderings of scenes by encoding the volumetric density and color of a scene within the weights of a coordinate-based multi-layer perceptron (MLP). This approach has enabled significant progress towards photorealistic view synthesis [30]. However, NeRF models the input to the MLP using infinitesimally small 3D points along a ray, which causes aliasing when rendering views of varying resolutions. Mip-NeRF rectified this problem by extending NeRF to instead reason about volumetric frustums along a cone [3]. Though this improves quality, both NeRF and mip-NeRF struggle when dealing with *unbounded* scenes, where the camera may face any direction and scene content may exist at any distance. In this work, we present an extension to mip-NeRF we call "mip-NeRF 360" that is capable of producing realistic renderings of these unbounded scenes, as shown in Figure 1.

Applying NeRF-like models to large unbounded scenes raises three critical issues:

1. Parameterization. Unbounded 360 degree scenes can occupy an arbitrarily large region of Euclidean space, but mip-NeRF requires that 3D scene coordinates lie in a bounded domain.
2. Efficiency. Large and detailed scenes require more network capacity, but densely querying a large MLP along each ray during training is expensive.
3. Ambiguity. The content of unbounded scenes may lie at any distance and will be observed by only a small number of rays, exacerbating the inherent ambiguity of reconstructing 3D content from 2D images.



(a) mip-NeRF [3], SSIM=0.526 (b) Our Model, SSIM=0.804

Figure 1. (a) Though mip-NeRF is able to produce accurate renderings of objects, for unbounded scenes it often generates blurry backgrounds and low-detail foregrounds. (b) Our model produces detailed realistic renderings of these unbounded scenes, as evidenced by the renderings (top) and depth maps (bottom) from both models. See the supplemental video for additional results.

Parameterization. Due to perspective projection, an object placed far from the camera will occupy a small portion of the image plane, but will occupy more of the image and be visible in detail if placed nearby. Therefore, an ideal parameterization of a 3D scene should allocate more capacity to nearby content and less capacity to distant content. Outside of NeRF, traditional view-synthesis methods address this by parameterizing the scene in projective panoramic

space [2, 4, 8, 14, 21, 24, 33, 42, 49] or by embedding scene content within some proxy geometry [15, 23, 38] that has been recovered using multi-view stereo.

One aspect of NeRF's success is its pairing of specific scene types with their appropriate 3D parameterizations. The original NeRF paper [30] focused on 360 degree captures of objects with masked backgrounds and on front-facing scenes where all images face roughly the same direction. For masked objects NeRF directly parameterized the scene in 3D Euclidean space, but for front-facing scenes NeRF used coordinates defined in projective space (normalized device coordinates, or "NDC" [5]). By warping an infinitely deep camera frustum into a bounded cube where distance along the z-axis corresponds to disparity (inverse distance), NDC effectively reallocates the NeRF MLP's capacity in a way that is consistent with the geometry of perspective projection.

However, scenes that are unbounded in *all* directions, not just in a single direction, require a different parameterization. This idea was explored by NeRF++ [46], which used an additional network to model distant objects, and by DONeRF [31], which proposed a space-warping procedure to shrink distant points towards the origin. Both of these approaches behave somewhat analogously to NDC but in *every* direction, rather than just along the z-axis. In this work, we extend this idea to mip-NeRF and present a method for applying any smooth parameterization to *volumes* (rather than points), and also present our own parameterization for unbounded scenes.

Efficiency. One fundamental challenge in dealing with unbounded scenes is that such scenes are often *large* and *detailed*. Though NeRF-like models can accurately reproduce objects or regions of scenes using a surprisingly small number of weights, the capacity of the NeRF MLP saturates when faced with increasingly intricate scene content. Additionally, larger scenes require significantly more samples along each ray to accurately localize surfaces. For example, when scaling NeRF from objects to buildings, Martin-Brualla *et al*. [27] doubled the number of MLP hidden units and increased the number of MLP evaluations by $8\times$. This increase in model capacity is expensive — a NeRF already takes multiple hours to train, and multiplying this time by an additional $\sim 40\times$ is prohibitively slow for most uses.

This training cost is exacerbated by the coarse-to-fine resampling strategy used by NeRF and mip-NeRF: MLPs are evaluated multiple times using "coarse" and "fine" ray intervals, and are supervised using an image reconstruction loss on both passes. This approach is wasteful, as the "coarse" rendering of the scene does not contribute to the final image. Instead of training a single NeRF MLP that is supervised at multiple scales, we will instead train two MLPs: a "proposal MLP" and a "NeRF MLP". The proposal MLP predicts volumetric density (but not color) and those densities are used to resample new intervals that are provided to the NeRF MLP, which then renders the image. Crucially, the weights produced by the proposal MLP are not supervised using the input image, but are instead supervised with the histogram weights generated by the NeRF MLP. This allows us to use a large NeRF MLP that is evaluated relatively few times, alongside a small proposal MLP that is evaluated many more times. As a result, our whole model's total capacity is significantly larger than mip-NeRF's ($\sim 15\times$), resulting in greatly improved rendering quality, but our training time only increases modestly ($\sim 2\times$).

We can think of this approach as a kind of "online distillation": while "distillation" commonly refers to training a small network to match the output of an already-trained large network [17], here we distill the structure of the outputs predicted by the NeRF MLP into the proposal MLP "online" by training both networks simultaneously. NeRV [43] performs a similar kind of online distillation for an entirely different task: training MLPs to approximate rendering integrals for the purpose of modeling visibility and indirect illumination. Our online distillation approach is similar in spirit to the "sampling oracle networks" used in DONeRF, though that approach uses ground-truth depth for supervision [31]. A related idea was used in TermiNeRF [36], though that approach only accelerates inference and actually *slows* training (a NeRF is trained to convergence, and an additional model is trained afterwards). A learned "proposer" network was explored in NeRF in Detail [1] but only achieves a speedup of 25%, while our approach accelerates training by 300%.

Several works have attempted to distill or "bake" a trained NeRF into a format that can be *rendered* quickly [16, 37, 45], but these techniques do not accelerate training. The idea of accelerating ray-tracing through a hierarchical data structure such as octrees [40] or bounding volume hierarchies [39] is well-explored in the rendering literature, though these approaches assume a-priori knowledge of the geometry of the scene and therefore do not naturally generalize to an inverse rendering context in which the geometry of the scene is unknown and must be recovered. Indeed, despite building an octree acceleration structure while optimizing a NeRF-like model, the Neural Sparse Voxel Fields approach does not significantly reduce training time [25].

Ambiguity. Though NeRFs are traditionally optimized using many input images of a scene, the problem of recovering a NeRF that produces realistic synthesized views from novel camera angles is still fundamentally underconstrained — an infinite family of NeRFs can explain away the input images, but only a small subset produces acceptable results for novel views. For example, a NeRF could recreate all input images by simply reconstructing each im-

age as a textured plane immediately in front of its respective camera. The original NeRF paper regularized ambiguous scenes by injecting Gaussian noise into the density head of the NeRF MLP before the rectifier [30], which encourages densities to gravitate towards either zero or infinity. Though this reduces some "floaters" by discouraging semi-transparent densities, we will show that it is insufficient for our more challenging task. Other regularizers for NeRF have been proposed, such as a robust loss on density [16] or smoothness penalties on surfaces [32, 48], but these solutions address different problems than ours (slow rendering and non-smooth surfaces, respectively). Additionally, these regularizers are designed for the point samples used by NeRF, while our approach is designed to work with the continuous weights defined along each mip-NeRF ray.

These three issues will be addressed in Sections 2, 3, and 4 respectively, after a review of mip-NeRF. We will demonstrate our improvement over prior work using a new dataset consisting of challenging indoor and outdoor scenes. We urge the reader to view our supplemental video, as our results are best appreciated when animated.

1. Preliminaries: mip-NeRF

Let us first describe how a fully-trained mip-NeRF [3] renders the color of a single ray cast into the scene $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ and $\mathbf{d}$ are the origin and direction of the ray respectively, and t denotes distance along the ray. In mip-NeRF, a sorted vector of distances $\mathbf{t}$ is defined and the ray is split into a set of intervals $T_i = [t_i, t_{i+1})$. For each interval i we compute the mean and covariance $(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathbf{r}(T_i)$ of the conical frustum corresponding to the interval (the radii of which are determined by the ray's focal length and pixel size on the image plane), and featurize those values using an integrated positional encoding:

$$\gamma(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \left\{ \begin{bmatrix} \sin(2^\ell \boldsymbol{\mu}) \exp\left(-2^{2\ell-1} \operatorname{diag}(\boldsymbol{\Sigma})\right) \\ \cos(2^\ell \boldsymbol{\mu}) \exp\left(-2^{2\ell-1} \operatorname{diag}(\boldsymbol{\Sigma})\right) \end{bmatrix} \right\}_{\ell=0}^{L-1} \quad (1)$$

This is the expectation of the encodings used by NeRF with respect to a Gaussian approximating the conical frustum. These features are used as input to an MLP parameterized by weights Θ_{NeRF} that outputs a density τ and color $\mathbf{c}$:

$$\forall T_i \in \mathbf{t}, \quad (\tau_i,\, \mathbf{c}_i) = \text{MLP}(\gamma(\mathbf{r}(T_i));\, \Theta_{\text{NeRF}})\,. \quad (2)$$

The view direction $\mathbf{d}$ is also provided as input to the MLP, but we omit this for simplicity. With these densities and colors we approximate the volume rendering integral using numerical quadrature [28]:

$$\mathbf{C}(\mathbf{r}, \mathbf{t}) = \sum_i w_i \mathbf{c}_i\,, \quad (3)$$

$$w_i = \left(1 - e^{-\tau_i (t_{i+1} - t_i)}\right) e^{-\sum_{i'<i} \tau_{i'} \left(t_{i'+1} - t_{i'}\right)} \quad (4)$$

where $\mathbf{C}(\mathbf{r}, \mathbf{t})$ is the final rendered pixel color. By construction, the alpha compositing weights $\mathbf{w}$ are guaranteed to sum to less than or equal to 1.

The ray is first rendered using evenly-spaced "coarse" distances $\mathbf{t}^c$, which are sorted samples from a uniform distribution spanning $[t_n, t_f]$, the camera's near and far planes:

$$t^c \sim \mathcal{U}[t_n, t_f]\,, \quad \mathbf{t}^c = \text{sort}(\{t^c\})\,. \quad (5)$$

During training this sampling is stochastic, but during evaluation samples are evenly spaced from t_n to t_f. After the MLP generates a vector of "coarse" weights $\mathbf{w}^c$, "fine" distances $\mathbf{t}^f$ are sampled from the histogram defined by $\mathbf{t}^c$ and $\mathbf{w}^c$ using inverse transform sampling:

$$t^f \sim \text{hist}(\mathbf{t}^c, \mathbf{w}^c)\,, \quad \mathbf{t}^f = \text{sort}(\{t^f\})\,. \quad (6)$$

Because the coarse weights $\mathbf{w}^c$ tend to concentrate around scene content, this strategy improves sampling efficiency.

A mip-NeRF is recovered by optimizing MLP parameters Θ_{NeRF} via gradient descent to minimize a weighted combination of coarse and fine reconstruction losses:

$$\sum_{\mathbf{r} \in \mathcal{R}} \frac{1}{10} \mathcal{L}_{\text{recon}}(\mathbf{C}(\mathbf{r}, \mathbf{t}^c), \mathbf{C}^*(\mathbf{r})) + \mathcal{L}_{\text{recon}}(\mathbf{C}(\mathbf{r}, \mathbf{t}^f), \mathbf{C}^*(\mathbf{r})) \quad (7)$$

where $\mathcal{R}$ is the set of rays in our training data, $\mathbf{C}^*(\mathbf{r})$ is the ground truth color corresponding to ray $\mathbf{r}$ taken from an input image, and $\mathcal{L}_{\text{recon}}$ is mean squared error.

2. Scene and Ray Parameterization

Though there exists prior work on the parameterization of *points* for unbounded scenes, this does not provide a solution for the mip-NeRF context, in which we must reparameterize *Gaussians*. To do this, first let us define $f(\mathbf{x})$ as some smooth coordinate transformation that maps from $\mathbb{R}^n \to \mathbb{R}^n$ (in our case, $n = 3$). We can compute the linear approximation of this function:

$$f(\mathbf{x}) \approx f(\boldsymbol{\mu}) + \mathbf{J}_f(\boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu}) \quad (8)$$

Where $\mathbf{J}_f(\boldsymbol{\mu})$ is the Jacobian of f at $\boldsymbol{\mu}$. With this, we can apply f to $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as follows:

$$f(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \left(f(\boldsymbol{\mu}),\, \mathbf{J}_f(\boldsymbol{\mu}) \boldsymbol{\Sigma} \mathbf{J}_f(\boldsymbol{\mu})^{\text{T}}\right) \quad (9)$$

This is functionally equivalent to the classic Extended Kalman filter [19], where f is the state transition model. Our choice for f is the following contraction:

$$\text{contract}(\mathbf{x}) = \begin{cases} \mathbf{x} & \|\mathbf{x}\| \leq 1 \\ \left(2 - \frac{1}{\|\mathbf{x}\|}\right)\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}\right) & \|\mathbf{x}\| > 1 \end{cases} \quad (10)$$

This design shares the same motivation as NDC: distant points should be distributed proportionally to disparity (inverse distance) rather than distance. In our model, instead

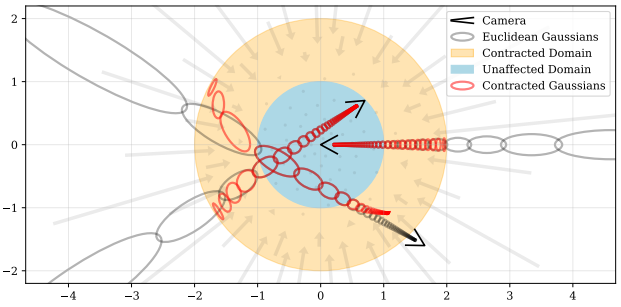


Figure 2. A 2D visualization of our scene parameterization. We define a $\mathrm{contract}(\cdot)$ operator (Equation 10, shown as arrows) that maps coordinates onto a ball of radius 2 (orange), where points within a radius of 1 (blue) are unaffected. We apply this contraction to mip-NeRF Gaussians in Euclidean 3D space (gray ellipses) similarly to a Kalman filter to produce our contracted Gaussians (red ellipses), whose centers are guaranteed to lie within a ball of radius 2. The design of $\mathrm{contract}(\cdot)$ combined with our choice to space ray intervals linearly according to disparity means that rays cast from a camera located at the origin of the scene will have equidistant intervals in the orange region, as demonstrated here.

of using mip-NeRF's IPE features in Euclidean space as per Equation 1 we use similar features (see supplement) in this contracted space: $\gamma(\mathrm{contract}(\boldsymbol{\mu}, \boldsymbol{\Sigma}))$. See Figure 2 for a visualization of this parameterization.

In addition to the question of how 3D coordinates should be parameterized, there is the question of how ray distances $\mathbf{t}$ should be selected. In NeRF this is usually done by sampling uniformly from the near and far plane as per Equation 5. However, if an NDC parameterization is used, this uniformly-spaced series of samples is actually uniformly spaced in *inverse* depth (disparity). This design decision is well-suited to unbounded scenes when the camera faces in only one direction, but is not applicable to scenes that are unbounded in all directions. We will therefore explicitly sample our distances $\mathbf{t}$ linearly in disparity (see [29] for a detailed motivation of this spacing).

To parameterize a ray in terms of disparity we define an invertible mapping between Euclidean ray distance t and a "normalized" ray distance s:

$$s \triangleq \frac{g(t) - g(t_n)}{g(t_f) - g(t_n)}, \quad t \triangleq g^{-1}(s \cdot g(t_f) + (1-s) \cdot g(t_n)), \quad (11)$$

where $g(\cdot)$ is some invertible scalar function. This gives us "normalized" ray distances $s \in [0, 1]$ that map to $[t_n, t_f]$. Throughout this paper we will refer to distances along a ray in either t-space or s-space, depending on which is more convenient or intuitive. By setting $g(x) = 1/x$ and constructing ray samples that are uniformly distributed in s-space, we produce ray samples whose t-distances are distributed linearly in disparity (additionally, setting $g(x) = \log(x)$ yields DONeRF's logarithmic spacing [31]). In our model, instead of performing the sampling in Equations 5 and 6 using t distances, we do so with s distances. This means that, not only are our initial samples spaced linearly in disparity, but subsequent resamplings from individual intervals of the weights $\mathbf{w}$ will also be distributed similarly. As can be seen from the camera in the center of Figure 2, this linear-in-disparity spacing of ray samples counter-balances $\mathrm{contract}(\cdot)$. Effectively, we have co-designed our scene coordinate space with our inverse-depth spacing, which gives us a parameterization of unbounded scenes that closely resembles the highly-effective setting of the original NeRF paper: evenly-spaced ray intervals within a bounded space.

3. Coarse-to-Fine Online Distillation

As discussed, mip-NeRF uses a coarse-to-fine resampling strategy (Figure 3) in which the MLP is evaluated once using "coarse" ray intervals and again using "fine" ray intervals, and is supervised using an image reconstruction loss at both levels. We instead train two MLPs, a "NeRF MLP" Θ_{NeRF} (which behaves similarly to the MLPs used by NeRF and mip-NeRF) and a "proposal MLP" Θ_{prop}. The proposal MLP predicts volumetric density, which is converted into a proposal weight vector $\hat{\mathbf{w}}$ according to Equation 4, but does not predict color. These proposal weights $\hat{\mathbf{w}}$ are used to sample s-intervals that are then provided to the NeRF MLP, which predicts its own weight vector $\mathbf{w}$ (and color estimates, for use in rendering an image). Critically, the proposal MLP is not trained to reproduce the

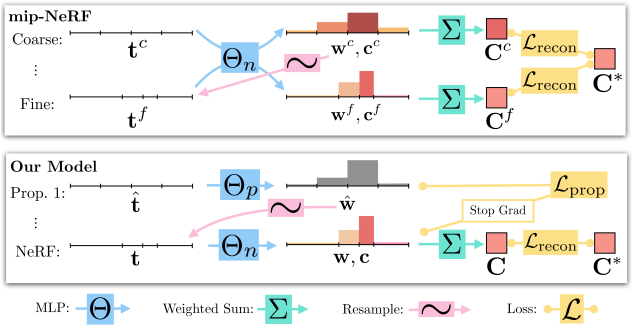


Figure 3. A comparison of our model's architecture with mip-NeRF's. Mip-NeRF uses one multi-scale MLP that is repeatedly queried (only two repetitions shown here) for weights that are resampled into intervals for the next stage, and supervises the renderings produced at all scales. We use a "proposal MLP" that emits weights (but not color) that are resampled, and in the final stage we use a "NeRF MLP" to produce weights and colors that result in the rendered image, which we supervise. The proposal MLP is trained to produce proposal weights $\hat{\mathbf{w}}$ that are consistent with the NeRF MLP's $\mathbf{w}$ output. By using a small proposal MLP and a large NeRF MLP we obtain a combined model with a high capacity that is still tractable to train.

input image, but is instead trained to bound the weights $\mathbf{w}$ produced by the NeRF MLP. Both MLPs are initialized randomly and trained jointly, so this supervision can be thought of as a kind of "online distillation" of the NeRF MLP's knowledge into the proposal MLP. We use a large NeRF MLP and a small proposal MLP, and repeatedly evaluate and resample from the proposal MLP with many samples (some figures and discussion illustrate only a single resampling for clarity) but evaluate the NeRF MLP only once with a smaller set of samples. This gives us a model that behaves as though it has a much higher capacity than mip-NeRF but is only moderately more expensive to train. As we will show, using a small MLP to model the proposal distribution does not reduce accuracy, which suggests that distilling the NeRF MLP is an easier task than view synthesis.

This online distillation requires a loss function that encourages the histograms emitted by the proposal MLP $(\hat{\mathbf{t}}, \hat{\mathbf{w}})$ and the NeRF MLP $(\mathbf{t}, \mathbf{w})$ to be consistent. At first this problem may seem trivial, as minimizing the dissimilarity between two histograms is a well-established task, but recall that the "bins" of those histograms $\mathbf{t}$ and $\hat{\mathbf{t}}$ need not be similar — indeed, if the proposal MLP successfully culls the set of distances where scene content exists, $\hat{\mathbf{t}}$ and $\mathbf{t}$ will be highly dissimilar. Though the literature contains numerous approaches for measuring the difference between two histograms with identical bins [11, 26, 35], our case is relatively underexplored. This problem is challenging because we cannot assume anything about the distribution of contents within one histogram bin: an interval with non-zero weight may indicate a uniform distribution of weight over that entire interval, a delta function located *anywhere* in that interval, or myriad other distributions. We therefore construct our loss under the following assumption: If it is *in any way possible* that both histograms can be explained using any single distribution of mass, then the loss must be zero. A non-zero loss can only be incurred if it is *impossible* that both histograms are reflections of the same "true" continuous underlying distribution of mass. See the supplement for visualizations of this concept.

To do this, we first define a function that computes the sum of all proposal weights that overlap with interval T:

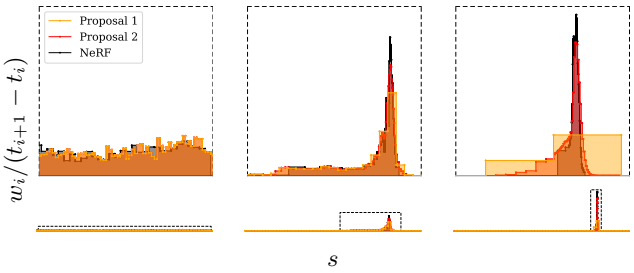
$$\text{bound}\big(\hat{\mathbf{t}}, \hat{\mathbf{w}}, T\big) = \sum_{j:\, T \cap \hat{T}_j \neq \varnothing} \hat{w}_j \,. \tag{12}$$

If the two histograms are consistent with each other, then it must hold that $w_i \leq \text{bound}\big(\hat{\mathbf{t}}, \hat{\mathbf{w}}, T_i\big)$ for all intervals (T_i, w_i) in $(\mathbf{t}, \mathbf{w})$. This property is similar to the additivity property of an outer measure in measure theory [13]. Our loss penalizes any surplus histogram mass that violates this inequality and exceeds this bound:

$$\mathcal{L}_{\text{prop}}\big(\mathbf{t}, \mathbf{w}, \hat{\mathbf{t}}, \hat{\mathbf{w}}\big) = \sum_i \frac{1}{w_i} \max\big(0, w_i - \text{bound}\big(\hat{\mathbf{t}}, \hat{\mathbf{w}}, T_i\big)\big)^2 \,, \tag{13}$$

This loss resembles a half-quadratic version of the chi-squared histogram distance that is often used in statistics and computer vision [35]. This loss is asymmetric because we only want to penalize the proposal weights for *underestimating* the distribution implied by the NeRF MLP — overestimates are to be expected, as the proposal weights will likely be more coarse than the NeRF weights, and will therefore form an upper envelope over it. The division by w_i guarantees that the gradient of this loss with respect to the bound is a constant value when the bound is zero, which leads to well-behaved optimization. Because $\mathbf{t}$ and $\hat{\mathbf{t}}$ are sorted, Equation 13 can be computed efficiently through the use of summed-area tables [10]. Note that this loss is invariant to monotonic transformations of distance t (assuming that $\mathbf{w}$ and $\hat{\mathbf{w}}$ have already been computed in t-space) so it behaves identically whether applied to Euclidean ray t-distances or to normalized ray s-distances.

We impose this loss between the NeRF histogram $(\mathbf{t}, \mathbf{w})$ and all proposal histograms $(\hat{\mathbf{t}}^k, \hat{\mathbf{w}}^k)$. The NeRF MLP is supervised using a reconstruction loss with the input image $\mathcal{L}_{\text{recon}}$, as in mip-NeRF. We place a stop-gradient on the NeRF MLP's outputs $\mathbf{t}$ and $\mathbf{w}$ when computing $\mathcal{L}_{\text{prop}}$ so that the NeRF MLP "leads" and the proposal MLP "follows" — otherwise the NeRF may be encouraged to produce a worse reconstruction of the scene so as to make the proposal MLP's job less difficult. The effect of this proposal supervision can be seen in Figure 4, where the NeRF MLP gradually localizes its weights $\mathbf{w}$ around a surface in the scene, while the proposal MLP "catches up" and predicts coarse proposal histograms that envelope the NeRF weights.



(a) 0% optimized (b) 4% optimized (c) 100% optimized

Figure 4. A visualization of the histograms $(\mathbf{t}, \mathbf{w})$ emitted from the NeRF MLP (black) and the two sets of histograms $(\hat{\mathbf{t}}, \hat{\mathbf{w}})$ emitted by the proposal MLP (yellow and orange) for a single ray from our dataset's *bicycle* scene over the course of training. Below we visualize the entire ray with fixed x and y axes, but above we crop both axes to better visualize details near scene content. Histogram weights are plotted as distributions that integrate to 1. (a) When training begins, all weights are uniformly distributed with respect to ray distance t. (b, c) As training progresses, the NeRF weights begin to concentrate around a surface and the proposal weights form a kind of envelope around those NeRF weights.

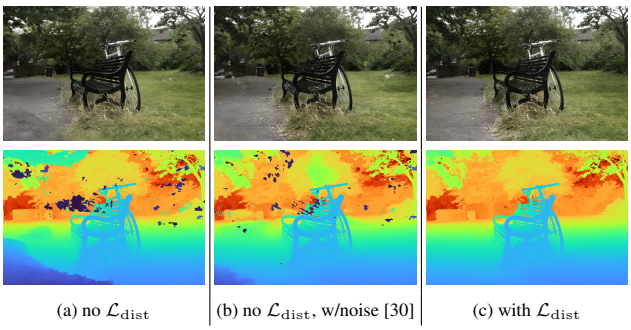
4. Regularization for Interval-Based Models

Due to ill-posedness, trained NeRFs often exhibit two characteristic artifacts we will call "floaters" and "background collapse", both shown in Figure 5(a). By "floaters" we refer to small disconnected regions of volumetrically dense space which serve to explain away some aspect of a subset of the input views, but when viewed from another angle look like blurry clouds. By "background collapse" we mean a phenomenon in which distant surfaces are incorrectly modeled as semi-transparent clouds of dense content close to the camera. Here we presents a regularizer that, as shown in Figure 5, prevents floaters and background collapse more effectively than the approach used by NeRF of injecting noise into volumetric density [30].

Our regularizer has a straightforward definition in terms of the step function defined by the set of (normalized) ray distances $\mathbf{s}$ and weights $\mathbf{w}$ that parameterize each ray:

$$\mathcal{L}_{\text{dist}}(\mathbf{s}, \mathbf{w}) = \iint_{-\infty}^{\infty} \mathbf{w}_{\mathbf{s}}(u)\mathbf{w}_{\mathbf{s}}(v)|u - v| \, d_u \, d_v \,, \qquad (14)$$

where $\mathbf{w}_{\mathbf{s}}(u)$ is interpolation into the step function defined by $(\mathbf{s}, \mathbf{w})$ at u: $\mathbf{w}_{\mathbf{s}}(u) = \sum_i w_i \mathbb{1}_{[s_i, s_{i+1})}(u)$. We use normalized ray distances $\mathbf{s}$ because using $\mathbf{t}$ significantly upweights distant intervals and causes nearby intervals to be effectively ignored. This loss is the integral of the distances between all pairs of points along this 1D step function, scaled by the weight w assigned to each point by the NeRF MLP. We refer to this as "distortion", as it resembles a continuous version of the distortion minimized by k-means (though it could also be thought of as maximizing a kind of autocorrelation). This loss is minimized by setting $\mathbf{w} = \mathbf{0}$ (recall that $\mathbf{w}$ sums to *no more than* 1, not exactly



(a) no $\mathcal{L}_{\text{dist}}$ (b) no $\mathcal{L}_{\text{dist}}$, w/noise [30] (c) with $\mathcal{L}_{\text{dist}}$

Figure 5. Our regularizer suppresses "floaters" (pieces of semi-transparent material floating in space, which are easy to identify in the depth map) and prevents a phenomenon in which surfaces in the background "collapse" towards the camera (shown in the bottom left of (a)). The noise-injection approach of Mildenhall *et al*. [30] only partially eliminates these artifacts, and reduces reconstruction quality (note the lack of detail in the depths of the distant trees). See the supplemental video for more visualizations.

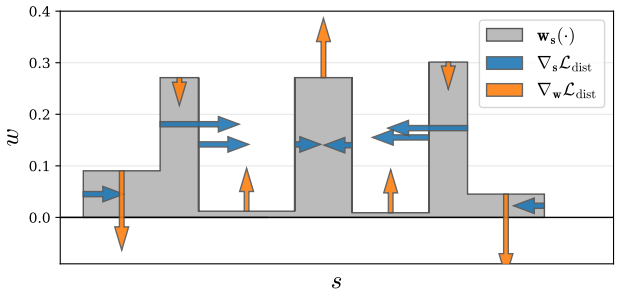


Figure 6. A visualization of $\nabla\mathcal{L}_{\text{dist}}$, the gradient of our regularizer, as a function of $\mathbf{s}$ and $\mathbf{w}$ on a toy step function. Our loss encourages each ray to be as compact as possible by 1) minimizing the width of each interval, 2) pulling distant intervals towards each other, 3) consolidating weight into a single interval or a small number of nearby intervals, and 4) driving all weights towards zero when possible (such as when the entire ray is unoccupied).

1). If that is not possible (i.e., if the ray is non-empty), it is minimized by consolidating weights into as small a region as possible. Figure 6 illustrates this behavior by showing the gradient of this loss on a toy histogram.

Though Equation 14 is straightforward to define, it is non-trivial to compute. But because $\mathbf{w}_{\mathbf{s}}(\cdot)$ has a constant value within each interval we can rewrite Equation 14 as:

$$\mathcal{L}_{\text{dist}}(\mathbf{s}, \mathbf{w}) = \sum_{i,j} w_i w_j \left| \frac{s_i + s_{i+1}}{2} - \frac{s_j + s_{j+1}}{2} \right| + \frac{1}{3} \sum_i w_i^2 (s_{i+1} - s_i) \qquad (15)$$

In this form, our distortion loss is trivial to compute. This reformulation also provides some intuition for how this loss behaves: the first term minimizes the weighted distances between all pairs of interval midpoints, and the second term minimizes the weighted size of each individual interval.

5. Optimization

Now that we have described our model components in general terms, we can detail the specific model used in all experiments. We use a proposal MLP with 4 layers and 256 hidden units and a NeRF MLP with 8 layers and 1024 hidden units, both of which use ReLU internal activations and a softplus activation for density τ. We do two stages of evaluation and resampling of the proposal MLP each using 64 samples to produce $(\hat{\mathbf{s}}^0, \hat{\mathbf{w}}^0)$ and $(\hat{\mathbf{s}}^1, \hat{\mathbf{w}}^1)$, and then one stage of evaluation of the NeRF MLP using 32 samples to produce $(\mathbf{s}, \mathbf{w})$. We minimize the following loss:

$$\mathcal{L}_{\text{recon}}(\mathbf{C}(\mathbf{t}), \mathbf{C}^*) + \lambda \mathcal{L}_{\text{dist}}(\mathbf{s}, \mathbf{w}) + \sum_{k=0}^{1} \mathcal{L}_{\text{prop}}(\mathbf{s}, \mathbf{w}, \hat{\mathbf{s}}^k, \hat{\mathbf{w}}^k) \,, \quad (16)$$

averaged over all rays in each batch (rays are not included in our notation). The λ hyperparameter balances our data

term $\mathcal{L}_{\text{recon}}$ and our regularizer $\mathcal{L}_{\text{dist}}$; we set $\lambda = 0.01$ in all experiments. The stop-gradient used in $\mathcal{L}_{\text{prop}}$ makes the optimization of Θ_{prop} independent from the optimization of Θ_{NeRF}, and as such there is no need for a hyperparameter to scale the effect of $\mathcal{L}_{\text{prop}}$. For $\mathcal{L}_{\text{recon}}$ we use Charbonnier loss [9]: $\sqrt{(x - x^*)^2 + \epsilon^2}$ with $\epsilon = 0.001$, which achieves slightly more stable optimization than the mean squared error used in mip-NeRF. We train our model (and all reported NeRF-like baselines) using a slightly modified version of mip-NeRF's learning schedule: 250k iterations of optimization with a batch size of 2^{14}, using Adam [22] with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$, a learning rate that is annealed log-linearly from 2×10^{-3} to 2×10^{-5} with a warm-up phase of 512 iterations, and gradient clipping to a norm of 10^{-3}.

6. Results

We evaluate our model on a novel dataset: 9 scenes (5 outdoors and 4 indoors) each containing a complex central object or area and a detailed background. During capture we attempted to prevent photometric variation by fixing camera exposure settings, minimizing lighting variation, and avoiding moving objects — we do not intend to probe all challenges presented by "in the wild" photo collections [27], only scale. Camera poses are estimated using COLMAP [41], as in NeRF. See the supplement for details.

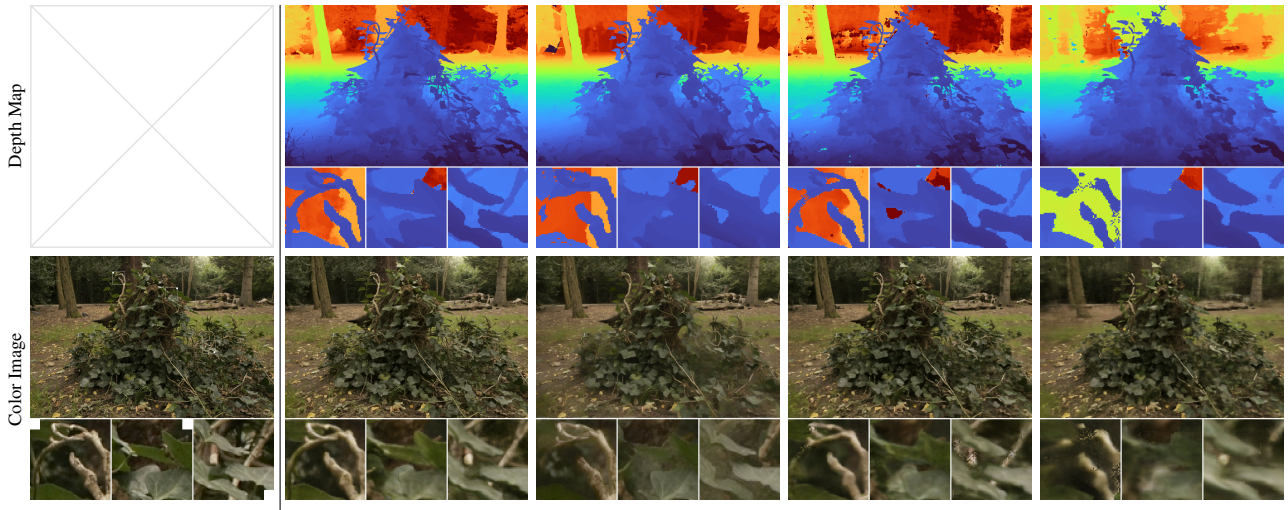
Compared methods. We compare our model with NeRF [30] and mip-NeRF [3], both using additional positional encoding frequencies so as to bound the entire scene inside the coordinate space used by both models. We evaluate against NeRF++ [46], which uses two MLPs to separately encode the "inside" and "outside" of each scene. We also evaluate against a version of NeRF that uses DONeRF's [31] scene parameterization, which uses logarithmically-spaced samples and a different contraction from our own. We also evaluate against mip-NeRF and NeRF++ variants in which the MLP(s) underlying each model have been scaled up to roughly match our own model in terms of number of parameter count (1024 hidden units for mip-NeRF, 512 hidden units for both MLPs in NeRF++). We evaluate against Stable View Synthesis [38], a non-NeRF model that represents the state-of-the-art of a different view-synthesis paradigm in which neural networks are trained on external scenes and combined with a proxy geometry produced by structure-from-motion [41]. We additionally compare with the publicly available SIBR implementations [7] of Deep Blending [15] and Point-Based Neural Rendering [23], two real-time IBR-based view synthesis approaches that also depend on an external proxy geometry. We also present a variant of our own model in which we use the latent appearance embedding (4 dimensions) presented in NeRF-W [6, 27] which ameliorates artifacts caused by inconsistent lighting conditions during scene capture (because our scenes do not contain transient objects, we do not benefit from NeRF-W's other components).

	PSNR ↑	SSIM ↑	LPIPS ↓	Time (hrs)	# Params
NeRF [12, 30]	23.85	0.605	0.451	4.16	1.5M
NeRF w/ DONeRF [31] param.	24.03	0.607	0.455	4.59	1.4M
mip-NeRF [3]	24.04	0.616	0.441	3.17	0.7M
NeRF++ [46]	25.11	0.676	0.375	9.45	2.4M
Deep Blending [15]	23.70	0.666	0.318	-	-
Point-Based Neural Rendering [23]	23.71	0.735	0.252	-	-
Stable View Synthesis [38]	25.33	0.771	0.211	-	-
mip-NeRF [3] w/bigger MLP	26.19	0.748	0.285	22.71	9.0M
NeRF++ [46] w/bigger MLPs	26.39	0.750	0.293	19.88	9.0M
Our Model	27.69	0.792	0.237	6.89	9.9M
Our Model w/GLO	26.26	0.786	0.237	6.90	9.9M

Table 1. A quantitative comparison of our model with several prior works using the dataset presented in this paper.

Comparative evaluation. In Table 1 we report mean PSNR, SSIM [44], and LPIPS [47] across the test images in our dataset. For all NeRF-like models, we report train times from a TPU v2 with 32 cores [18], as well as model size (the train times and model sizes of SVS, Deep Blending, and Point-Based Neural Rendering are not presented, as this comparison would not be particularly meaningful). Our model outperforms all prior NeRF-like models by a significant margin, and we see a 57% reduction in mean squared error relative to mip-NeRF with a $2.17\times$ increase in train time. The mip-NeRF and NeRF++ baselines that use larger MLPs are more competitive, but are $\sim 3\times$ slower to train than our model and still achieve significantly lower accuracies. Our model outperforms Deep Blending and Point-Based Neural Rendering across all error metrics. It also outperforms SVS for PSNR and SSIM, but not LPIPS. This may be due to SVS being supervised to directly minimize an LPIPS-like perceptual loss, while we minimize a per-pixel reconstruction loss. See the supplement for renderings from SVS that achieve lower LPIPS scores than our model despite having reduced image quality [20]. Our model has several advantages over SVS and Deep Blending in addition to image quality: those models require external training data while our model does not, those models require the proxy geometry produced by a MVS package (and may fail when that geometry is incorrect) while we do not, and our model produces extremely detailed depth maps while SVS and Deep Blending do not (the "SVS depths" we show were produced by COLMAP [41] and are used as input to the model). Figure 7 shows model outputs, though we encourage the reader to view our supplemental video.

Ablation study. In Table 2 we present an ablation study of our model on the *bicycle* scene in our dataset, the findings of which we summarize here. A) Removing $\mathcal{L}_{\text{prop}}$ significantly reduces performance, as the proposal MLP is not



(a) Ground-Truth Test-Set Image (b) Our Model, SSIM=0.738 (c) SVS [38, 41], SSIM=0.680 (d) NeRF++ [46], SSIM=0.622 (e) mip-NeRF [3], SSIM=0.522

Figure 7. (a) A test-set image from our dataset's *stump* scene, with (b) our model's rendered image and depth map (median ray termination distance [34]). Cropped patches are shown to highlight details. Compared to prior work (c-e) our renderings more closely resemble the ground-truth and our depths look more plausible (though no ground-truth depth is available). See the supplement for more results.

supervised during training. B) Removing $\mathcal{L}_{\text{dist}}$ does not substantially affect our metrics but results in "floater" artifacts in scene geometry, as shown in Figure 5. C) the regularization proposed by Mildenhall *et al*. [30] of injecting Gaussian noise ($\sigma = 1$) into density degrades performance (and as shown in Figure 5 is less effective at eliminating floaters). D) Removing the proposal MLP and using a single MLP to model both the scene and the proposal weights does not degrade performance but increases training time by $\sim 3\times$, hence our small proposal MLP. E) Removing the proposal MLP and training our model using mip-NeRF's approach (applying $\mathcal{L}_{\text{recon}}$ at all coarse scales instead of using our $\mathcal{L}_{\text{prop}}$) worsens both speed *and* accuracy, justifying our supervision strategy. F) Using a small NeRF MLP (256 hidden units instead of our 1024 hidden units) accelerates training but reduces quality, demonstrating the value of a high-capacity model when dealing with detailed scenes. G) Removing IPE completely and using NeRF's positional encoding [30] reduces performance, showing the value in building upon mip-NeRF instead of NeRF. H) Ablating the contraction and instead adding positional encoding frequencies to bound the scene decreases accuracy and speed. I) Using the parameterization and logarithmic ray-spacing presented in DONeRF [31] reduces accuracy.

	PSNR ↑	SSIM ↑	LPIPS ↓	Time (hrs)	# Params
A) No $\mathcal{L}_{\text{prop}}$	20.49	0.406	0.573	6.21	9.0M
B) No $\mathcal{L}_{\text{dist}}$	24.41	0.687	0.300	7.08	9.0M
C) No $\mathcal{L}_{\text{dist}}$, w/Noise Injection	24.00	0.655	0.328	7.08	9.0M
D) No Proposal MLP	24.26	0.682	0.307	18.89	8.7M
E) No Prop. MLP w/[3]'s Training	23.45	0.659	0.328	18.89	8.7M
F) Small NeRF MLP	22.80	0.515	0.480	4.31	1.1M
G) No IPE	23.87	0.664	0.322	7.08	9.0M
H) No Contraction	23.77	0.642	0.347	8.79	10.9M
I) w/DONeRFs Contraction [31]	23.99	0.654	0.334	7.20	9.0M
Our Complete Model	24.37	0.687	0.300	7.09	9.0M

Table 2. An ablation study in which we remove or replace model components to measure their effect. See the text for details.

Limitations. Though mip-NeRF 360 significantly outperforms mip-NeRF and other prior work, it is not perfect. Some thin structures and fine details may be missed, such as the tire spokes in the *bicycle* scene (Figure 5), or the veins on the leaves in the *stump* scene (Figure 7). View synthesis quality will likely degrade if the camera is moved far from the center of the scene. And, like most NeRF-like models, recovering a scene requires several hours of training on an accelerator, precluding on-device training.

7. Conclusion

We have presented mip-NeRF 360, a mip-NeRF extension designed for real-world scenes with unconstrained camera orientations. Using a novel Kalman-like scene parameterization, an efficient proposal-based coarse-to-fine distillation framework, and a regularizer designed for mip-NeRF ray intervals, we are able to synthesize realistic novel views and complex depth maps for challenging unbounded real-world scenes, with a 57% reduction in mean-squared error compared to mip-NeRF.

Acknowledgements. Our sincere thanks to David Salesin and Ricardo Martin-Brualla for their help in reviewing this paper before submission, and to George Drettakis and Georgios Kopanas for their help in evaluating baselines on our 360 dataset.

References

[1] Relja Arandjelović and Andrew Zisserman. NeRF in detail: Learning to sample for view synthesis. *arXiv:2106.05264*, 2021. 2

[2] Benjamin Attal, Selena Ling, Aaron Gokaslan, Christian Richardt, and James Tompkin. MatryODShka: Real-time 6DoF video view synthesis using multi-sphere images. *ECCV*, 2020. 2

[3] 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. 1, 3, 7, 8

[4] Tobias Bertel, Mingze Yuan, Reuben Lindroos, and Christian Richardt. OmniPhotos: Casual 360° VR photography. *ACM Transactions on Graphics*, 2020. 2

[5] J.F. Blinn. A trip down the graphics pipeline: pixel coordinates. *IEEE Computer Graphics and Applications*, 1991. 2

[6] Piotr Bojanowski, Armand Joulin, David Lopez-Pas, and Arthur Szlam. Optimizing the latent space of generative networks. *ICML*, 2018. 7

[7] Sebastien Bonopera, Peter Hedman, Jerome Esnault, Siddhant Prakash, Simon Rodriguez, Theo Thonat, Mehdi Benadel, Gaurav Chaurasia, Julien Philip, and George Drettakis. sibr: A system for image based rendering, 2020. 7

[8] Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew DuVall, Jason Dourgarian, Jay Busch, Matt Whalen, and Paul Debevec. Immersive light field video with a layered mesh representation. *SIGGRAPH*, 2020. 2

[9] Pierre Charbonnier, Laure Blanc-Feraud, Gilles Aubert, and Michel Barlaud. Two deterministic half-quadratic regularization algorithms for computed imaging. *International Conference on Image Processing*, 1994. 7

[10] Franklin C Crow. Summed-area tables for texture mapping. *SIGGRAPH*, 1984. 5

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *CVPR*, 2005. 5

[12] 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`. 7

[13] Lawrence C Evans and Ronald F Garzepy. *Measure theory and fine properties of functions*. Routledge, 2018. 5

[14] Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. Casual 3D Photography. *SIGGRAPH Asia*, 2017. 2

[15] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *SIGGRAPH Asia*, 2018. 2, 7

[16] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021. 2, 3

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015. 2

[18] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. *International Symposium on Computer Architecture*, 2017. 7

[19] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960. 3

[20] Markus Kettunen, Erik Härkönen, and Jaakko Lehtinen. E-lpips: robust perceptual image similarity via random transformation ensembles. *arXiv:1906.03973*, 2019. 7

[21] Wesley Khademi and Jonathan Ventura. View synthesis in casually captured scenes using a cylindrical neural radiance field with exposure compensation. *ACM SIGGRAPH 2021 Posters*, 2021. 2

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

[23] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. *Computer Graphics Forum*, 2021. 2, 7

[24] Kai-En Lin, Zexiang Xu, Ben Mildenhall, Pratul P Srinivasan, Yannick Hold-Geoffroy, Stephen DiVerdi, Qi Sun, Kalyan Sunkavalli, and Ravi Ramamoorthi. Deep multi depth panoramas for view synthesis. *ECCV*, 2020. 2

[25] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 2

[26] Subhransu Maji, Alexander C Berg, and Jitendra Malik. Classification using intersection kernel support vector machines is efficient. *CVPR*, 2008. 5

[27] 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. 2, 7

[28] Nelson Max. Optical models for direct volume rendering. *IEEE TVCG*, 1995. 3

[29] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Transactions on Graphics (TOG)*, 2019. 4

[30] 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. 1, 2, 3, 6, 7, 8

[31] 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. 2, 4, 7, 8

[32] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. *ICCV*, 2021. 3

[33] Ryan S Overbeck, Daniel Erickson, Daniel Evangelakos, Matt Pharr, and Paul Debevec. A system for acquiring, processing, and rendering panoramic light field stills for virtual reality. *ACM Transactions on Graphics*, 2018. 2

[34] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable Neural Radiance Fields. *ICCV*, 2021. 8

[35] Ofir Pele and Michael Werman. The quadratic-chi histogram distance family. *ECCV*, 2010. 5

[36] Martin Piala and Ronald Clark. Terminerf: Ray termination prediction for efficient neural rendering. *3DV*, 2021. 2

[37] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *ICCV*, 2021. 2

[38] Gernot Riegler and Vladlen Koltun. Stable view synthesis. *CVPR*, 2021. 2, 7, 8

[39] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. *SIGGRAPH*, 1980. 2

[40] Hanan Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1990. 2

[41] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *CVPR*, 2016. 7, 8

[42] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. *SIGGRAPH*, 1999. 2

[43] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. NeRV: Neural reflectance and visibility fields for relighting and view synthesis. *CVPR*, 2021. 2

[44] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE TIP*, 2004. 7

[45] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. *ICCV*, 2021. 2

[46] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492*, 2020. 2, 7, 8

[47] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CVPR*, 2018. 7

[48] Xiuming Zhang, Pratul P. Srinivasan, Boyang Deng, Paul Debevec, William T. Freeman, and Jonathan T. Barron. NeRFactor: Neural Factorization of Shape and Reflectance Under an Unknown Illumination. *SIGGRAPH Asia*, 2021. 3

[49] Ke Colin Zheng, Sing Bing Kang, Michael F Cohen, and Richard Szeliski. Layered depth panoramas. *CVPR*, 2007. 2