Eulerian Gaussian Splatting using Hashed Probability Pyramids

Mia Gaia Polansky[1] George Kopanas[2]
Stephan Garbin[3] Todd Zickler[1] Dor Verbin[2]

[1]Harvard University [2]Google DeepMind [3]Google

Abstract

We introduce a probabilistic splat-based radiance field framework that retains the fast rasterization and test-time efficiency of 3D Gaussian Splatting (3DGS) while replacing heuristic primitive manipulation with gradient-based optimization of a volumetric probability density. Rather than relocating, splitting, or culling Gaussians via hand-tuned densification (e.g., ADC), we treat primitive locations as samples drawn from a persistent, learnable density. We instantiate this density with a novel, memory-efficient multi-scale hierarchical grid that enables end-to-end gradient-based control over primitive population density. To stabilize stochastic training, we derive an unbiased gradient estimator with control variates that markedly reduces variance. By allowing probability mass to flow to where the loss demands, our method eliminates brittle priors and naturally explores the volume, achieving state-of-the-art reconstruction quality on mip-NeRF 360 while preserving 3DGS-level rendering speed.

1. Introduction

Neural radiance fields (NeRFs) [17] achieve high quality novel view synthesis by optimizing a continuous volumetric density using gradient descent. 3D Gaussian splatting (3DGS) [11] accelerates rendering by representing scenes with discrete primitives, but it relies on sometimes-brittle heuristics for adding and removing primitives during optimization.

We introduce Eulerian Gaussian splatting (EGS), an Eulerian formulation of splat-based view synthesis that bridges these two approaches, combining the stability of continuous-field optimization with the runtime efficiency of discrete Gaussian primitives. We do this by defining a volumetric probability density function over the scene, and optimizing this field with gradient descent by rendering a discrete set of Gaussians sampled from the field at each iteration.

In contrast to 3DGS and variants such as 3DGS-MCMC [12], which take a Lagrangian view by explicitly relocating and restructuring primitives, EGS adopts an Eulerian perspective and optimizes an underlying probability function that governs where Gaussians are allocated. This allows new spatial structures to emerge wherever the loss demands, exploring the volume purely through gradient descent, without any hand-crafted procedures for moving, splitting, or pruning primitives. It bridges the gap between NeRF and 3DGS by producing an efficient 3DGS-style representation at render time, while retaining NeRF-like flexibility to adaptively allocate probability mass during optimization.

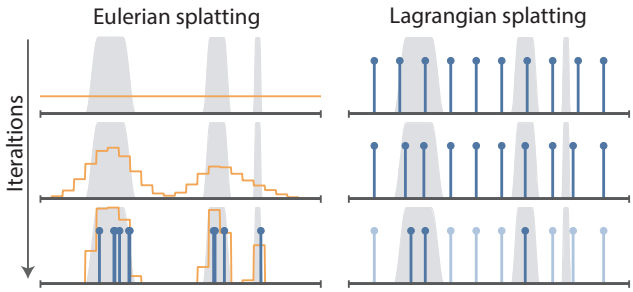

Figure 1. *Left*: We fit to scene shape (gray) by optimizing a probability distribution (orange) that governs primitive locations, creating and removing mass wherever needed, and producing primitives (blue stems) via sampling. *Right*: Previous methods directly optimize primitives, and primitives with no local gradient cannot adapt without additional heuristics for erasure and re-insertion.

To realize our formulation, we address several challenges. We introduce the *hashed probability pyramid*, a novel volumetric probability representation that achieves high resolution while being memory-efficient, globally normalized, and amenable to efficient sampling. The key idea is to represent the probability density using a multi-scale hierarchical grid, and to exploit scene sparsity by sharing parameters at finer levels of the hierarchy. This allows for resolving thin structures while also being efficient enough for end-to-end optimization on a single GPU.

Optimizing our probability pyramid with stochastic gradient descent introduces another challenge: constructing a gradient estimator with sufficiently low variance to ensure stable learning. We address this by using control variates to

derive an unbiased estimator that isolates each Gaussian's contribution to the rendered images, yielding significantly more informative and less noisy gradient updates.

We experimentally show that our method outperforms all baselines when trained with the same number of primitives initialized from a uniform distribution over the scene volume and achieves results comparable to state-of-the-art methods that depend on COLMAP-based initialization.

2. Related Work

Radiance fields and hash-encoding grids. Radiance fields have become a dominant approach for high-quality novel view synthesis from posed images. Neural radiance fields (NeRFs) [1, 17] model a scene as a continuous volumetric function parameterized by an MLP and optimized end-to-end via differentiable rendering. This has the advantage of allowing optimization by pure gradient decent. However, rendering novel views is slow and computationally-intensive because it requires evaluating MLPs at millions of sample points per image.

In response, grid-based radiance fields have gained favor because they replace expensive per-sample MLP evaluations with cache-friendly look-ups of precomputed features, enabling orders-of-magnitude speedups in training and rendering [10, 21]. The main drawback of this is memory, which scales quickly with the number of voxels in which features are stored, making high-fidelity reconstructions intractable.

To alleviate memory constraints, sparse hierarchical structures [24, 25] and hash encoding grids [2, 19] encode high dimensional feature grids in a compact and scalable manner by compressing dense grids using shared feature tables that are indexed by a hash function. We use existing hash grids to store the attributes of Gaussian primitives (color, scale, *etc.*), and we create a new hashed hierarchical representation for volumetric probability density that is globally normalized.

Adaptive density control for Gaussian splatting. Another way to perform view synthesis is using discrete primitives. This trades some representational flexibility for explicit control and real-time, hardware-friendly rasterization. 3D Gaussian Splatting (3DGS) represents a scene with a compact set of anisotropic Gaussian primitives and uses differentiable rasterization for real-time rendering [11]. Its training typically relies on heuristics called "Adaptive Density Control" (ADC) to manipulate primitives, for example using the magnitude of positional gradients to identify regions that need more primitives. Similar approaches are used for splatting other methods based on Lagrangian scene representations [6, 9, 15].

There are efforts to make density control more principled. Taming-3DGS [16] replaces hand-tuning with a score-based procedure that ranks Gaussians using per-pixel saliency, positional gradients, and primitive attributes. Revising Densification [3] introduces per-pixel error signals and opacity-aware cloning. Efficient Density Control [4] proposes adaptive rules for splitting and pruning. However, these methods still rely on schedules and thresholds that can be brittle.

Probabilistic approaches to novel view synthesis. Challenges with rule-based density control motivates a shift to probabilistic ways of guiding primitives during training. For example, MCMC-3DGS [12] encourages high-transparency primitives to randomly explore space using Brownian motion and random re-spawn mechanics, while being guided by the probability distribution that is implied by the positions of opaque primitives. This approach has been extended to non-Gaussian primitives as well [14]. Our method is different because it learns a probability distribution explicitly, eliminating the need for density control altogether.

Implicit Neural Point Clouds (INPC) [7] introduces an explicit probability distribution for placing point primitives in a scene. However, their probability field is based on octrees, so its spatial refinement still relies on heuristic subdivision and pruning rules, which can limit flexibility and introduce suboptimal structural decisions.

Another adjacent method is Lagrangian Hashing [5], which compresses neural fields by replacing high-resolution hash encoding grids with a point-based "Langrangian" representation stored in the upper levels of a hash table. Conceptually, this bridges Eulerian grids and point-based primitives, but unlike our approach it does not learn a normalized spatial probability distribution over primitive locations; instead, capacity is redistributed by an error-driven objective.

3. Method

The input to our method is a dataset of images of a scene with their corresponding known cameras, and our goal is to optimize the scene to match the input images. Our rendering model is probabilistic:

$$I = \mathbb{E}_{\mathcal{G}\sim p(\mathcal{G})}\left[\text{Render}(\mathcal{G}, \pi)\right], \tag{1}$$

where $\mathcal{G} = \{\mathcal{G}_i\}_{i=0}^{N-1}$ is a set of N Gaussians sampled from distribution $p(\mathcal{G})$, and $\text{Render}(\cdot)$ is the standard 3DGS rasterization which renders the Gaussians from camera π.

This probabilistic rendering model is the crux of our approach: optimizing p allows our model to efficiently create and remove Gaussians throughout the scene using only gradient cues, while using efficient rasterization for optimization.

In Sec. 3.1 we describe our probabilistic model for p, and in Sec. 3.2 we introduce our novel parameterization and sampling procedure. In Sec. 4, we describe how to optimize this new probability data structure.

3.1. Probabilistic Model

Typical scenes reconstructed by methods based on Gaussian splatting require millions of Gaussians, each of which

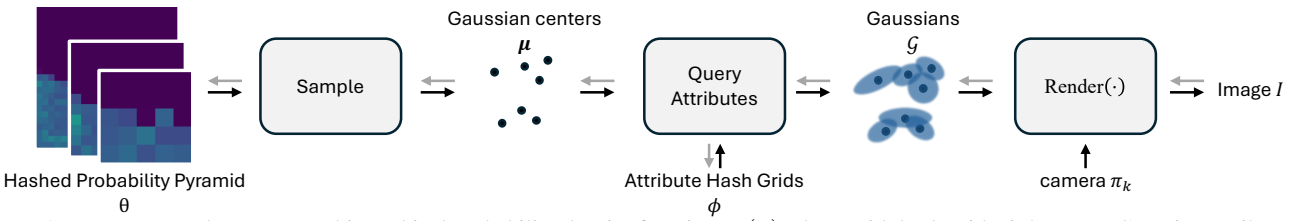

Figure 2. We represent the scene as a hierarchical probability density function $p_\theta(\mu)$, along with hash grids ϕ that store Gaussian attributes (color, opacity, scale, rotation) at each position. At each training iteration, we sample Gaussian centers $\mu_i \sim p_\theta(\mu)$, look up their attributes $\phi(\mu_i)$, and render the resulting Gaussians with a training camera π_k to create image I (black arrows). The gradient of the loss between rendered image I and the corresponding training image is used to update parameters θ and ϕ (gray arrows).

comprises tens of scalars for the Gaussian's position, scale, rotation, opacity, and (view-dependent) color. Modeling and sampling the full distribution p in Eq. (1) is prohibitively expensive, so we make some simplifying assumptions in modeling it.

First, we assume that the Gaussians are independent, so:

$$p(\mathcal{G}) = \prod_{i=0}^{N-1} p(\mathcal{G}_i), \tag{2}$$

where $p(\mathcal{G}_i)$ is the probability distribution function of a single Gaussian. Second, we split the probability distribution of each Gaussian into a distribution with parameters θ over the position of the Gaussian, and a distribution with parameters ϕ over the remaining Gaussian's attributes:

$$p(\mathcal{G}_i) = p_\theta(\mu_i) p_\phi(\phi_i | \mu_i), \tag{3}$$

where μ_i is the mean of the ith Gaussian, and ϕ_i are its remaining attributes (rotation, scale, opacity, and color).

Since the parameters of the attributes ϕ are significantly easier to optimize compared with the Gaussian means, we use a deterministic (optimizable) model for $p_\phi(\phi_i|\mu_i)$, realized as a 3D hash grid [18]. The parameters of the hash grid ϕ are used to map a Gaussian position μ_i to its attributes ϕ_i.

3.2. Hashed Probability Pyramid

We need a probability distribution over 3D space, say $[0,1]^3$, that has two key properties: (i) it can be sampled efficiently, and (ii) it supports high resolutions without an infeasible growth in parameters. We achieve this by introducing a *hashed probability pyramid*, which is a multiscale piecewise-constant distribution function in 3D. Piecewise-constant distributions have been widely used in computer graphics for sampling in one or two dimensions [20], and our model extends their benefits to 3D without a cubic growth in memory.

We parameterize the full distribution as a product of functions over L levels:

$$p_\theta(\mu) = \frac{1}{Z} \prod_{\ell=0}^{L-1} f_\theta^{(\ell)}(\mu), \tag{4}$$

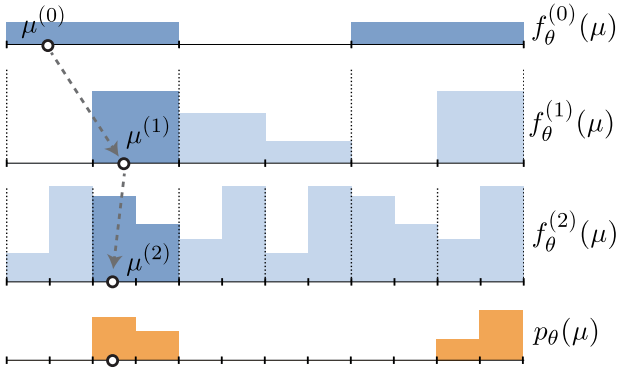

Figure 3. Sampling from a hashed probability pyramid with $L = 3$ levels and budget $B = 3$. Levels 1 and 2 are normalized within each 2-bin block. Level 2 is hashed: Blocks 1–3 are the same as blocks 4–6. Thus, the pyramid mimics a 12-bin distribution $p_\theta(\mu)$ using only eight degrees of freedom.

where Z is a normalizing factor. Each $f_\theta^{(\ell)}$ is a piecewise-constant function with $N_\ell \times N_\ell \times N_\ell$ bins that partition the volume:

$$f_\theta^{(\ell)}(\mu) = \sum_{\nu \in \mathcal{I}_\ell} \theta_\nu^{(\ell)} \mathbb{1}\left[\lfloor \mu N_\ell \rfloor = \nu\right], \tag{5}$$

where $\mathcal{I}_\ell = \{0, ..., N_\ell - 1\}^3$ is the set of 3D indices of the ℓth level grid, and $\theta_\nu^{(\ell)}$ is a parameter representing the unnormalized probability density associated with index ν in level ℓ. The expression $\mathbb{1}\left[\lfloor \mu N_\ell \rfloor = \nu\right]$ partitions the volume $[0,1]^3$ into bins: it returns 1 if μ is in the bin with index ν and 0 otherwise. In our pyramid, level $\ell = 0$ is the coarsest (lowest resolution), and level $\ell = L - 1$ is finest (highest resolution). We assume for simplicity that resolutions grow by a factor of two, *i.e.*, $N_{\ell+1} = 2N_\ell$.

The 0th function $f_\theta^{(0)}$ is normalized to integrate to 1, and the other functions $f_\theta^{(1)}, ..., f_\theta^{(L-1)}$ are normalized such that each of their $2 \times 2 \times 2$ blocks integrates to 1. This makes the parameterization *complete*, meaning that any $p(\mu)$ can be written as a product of such functions without introducing additional degrees of freedom. We see this by tallying the

pyramid's degrees of freedom,

$$N_0^3 - 1 + \sum_{\ell=1}^{L-1} ((N_{\ell+1}/N_\ell)^3 - 1) \cdot N_\ell^3 = N_{L-1}^3 - 1, \quad (6)$$

which is the same as that of a standard distribution with $N_{L-1} \times N_{L-1} \times N_{L-1}$ bins.

However, naïvely using $N_{L-1}^3 - 1$ parameters to represent our distribution is infeasible for high resolution grids. To address this, we upper bound the number of parameters in each level. We allocate at most B distinct blocks that are $2 \times 2 \times 2$ at each level, and we randomly tile them across the level's bins using a hash function $T_\ell \colon \{0, ..., N_\ell - 1\}^3 \to \{0, ..., B - 1\}$. This populates the $N_\ell \times N_\ell \times N_\ell$ bins at the level using only B distributions, with the bin whose index is $\nu \in \{0, ..., N_{\ell-1}\}^3$ being populated by a value from the $2 \times 2 \times 2$ block given by $T_\ell(\nu)$. Fig. 3 shows an example in one dimension.

Hashing greatly reduces the parameter count without significantly impacting representational capacity. There can be "hash collisions" for pyramid levels with $N_\ell > 2B^{1/3}$, meaning that multiple bins from level $\ell - l$ are mapped to the same block at level ℓ; but the sparsity of surfaces in three dimensions means that these collisions often occur in empty parts of the volume which, as depicted in Fig. 3, can easily be accommodated by setting coarser bins to zero. Note that hash collisions are independent for each level, meaning that the probability that a certain level has the same collision in all levels is vanishingly small. These are similar to the motivations for Instant-NGP [19], and in fact we use their hash encoding for our T_ℓ maps. See the supplement for details.

3.2.1. Sampling from a Hashed Probability Pyramid

Another requirement of our representation is efficient sampling. This is important for optimizing the distribution parameters θ and Gaussian attributes ϕ based on the probabilistic rendering model in Eq. (1). Fortunately, sampling from the hashed probability pyramid is trivial: We sample from the coarsest distribution and then iteratively refine the sample by traversing levels conditioned on the previous bin.

Formally, we generate a sample $\mu^{(L-1)} \sim p_\theta(\mu)$ by:

$$\begin{aligned} &\mu^{(0)} \sim p_\theta^{(0)}(\mu), \quad (7) \\ &\mu^{(\ell)} \sim p_\theta^{(\ell)}\left(\mu \,\middle|\, \lfloor \mu^{(\ell-1)} N_{i-1} \rfloor\right), \text{ for } \ell = 1, ..., L-1, \end{aligned}$$

where $p_\theta^{(0)}(\mu) = \frac{1}{Z_0} f_\theta^{(0)}(\mu)$ is the distribution corresponding to the coarsest level of the pyramid in Eq. (4) (Z_0 is the normalization factor), and the ℓth conditional distribution $p_\theta^{(\ell)}(\mu|\cdot)$ for $\ell = 1, ..., L-1$ is the $2 \times 2 \times 2$ piecewise-constant distribution that subdivides the bin that was sampled at level $\ell - 1$. See Fig. 3.

Each of the sampling stages in Eq. (7) can be done efficiently with standard inverse transform sampling, which passes a three-vector $u^{(\ell)} \sim \mathrm{U}([0,1]^3)$ through the appropriate inverse cumulative distribution function for each stage. Also, we make the sequence of L sampling stages end-to-end auto-differentiable by tying the stages together via: (i) drawing a single random sample $u^{(0)} \sim \mathrm{U}([0,1]^3)$; and then (ii) recursively defining subsequent three-vectors using deterministic maps:

$$u^{(\ell)} = \mathrm{frac}\left(\mu^{(\ell-1)} N_{\ell-1}\right) \text{ for } \ell = 1, ..., L-1, \quad (8)$$

where $\mathrm{frac}(x) = x - \lfloor x \rfloor$ is the fractional part function. This works because all of our distributions are piecewise-constant, implying that the distribution within each bin is uniform, and thus that Eq. (8) yields valid samples from $\mathrm{U}([0,1]^3)$ for all levels ℓ. This approach guarantees the differentiability of the final sample position $\mu^{(L-1)}$ with respect to all of the pyramid parameters that led to it.

3.2.2. Handling Unbounded Scenes

Our hashed probability pyramid provides samples in $[0,1]^3$, but 3D scenes are often unbounded. In order to map the samples to all of 3D space $\mathbb{R}^3$, we first apply an affine mapping to map our samples to $[-1,1]^3$ and then we use a contraction function $\mathcal{C} : [-1,1]^3 \to \mathbb{R}^3$ defined as:

$$\mathcal{C}(\mu) = \begin{cases} \frac{\mu}{a}, & \|\mu\|_\infty \le a, \\ \frac{1-a}{1-\|\mu\|_\infty} \frac{\mu}{\|\mu\|_\infty}, & \text{otherwise}, \end{cases} \quad (9)$$

where $\|\mu\|_\infty = \max_n |\mu_n|$ denotes the L_∞ norm, and a is a contraction factor that controls how the model's capacity is allocated to near and far content. (We set $a = 3/4$ in our experiments.)

Our contraction function is a modification of the (inverse) contraction function from mip-NeRF 360 [1], but modified to have L_∞ instead of L_2 symmetry, since our distribution is defined using a cubic grid.

4. Optimization

Our optimization task is to adjust distribution parameters θ and Gaussian attributes ϕ to minimize the error between a set of observed images $\{I_k^{\mathrm{GT}}\}_{k=0}^{K-1}$ and the corresponding renderings $\{I_k\}_{k=0}^{K-1}$ as described in Sec. 3. The loss can be written as

$$\sum_{k=0}^{K-1} \mathcal{L}(I_k, I_k^{\mathrm{GT}}), \quad (10)$$

where $\mathcal{L}$ is the loss between one pair of images.

Minimizing the loss by gradient descent requires estimating the gradient with respect to $\xi \in \{\theta, \phi\}$. We write:

$$\begin{aligned} &\nabla_\xi \mathcal{L}(I, I^{\mathrm{GT}}) = \nabla_I \mathcal{L}(I, I^{\mathrm{GT}}) \cdot \nabla_\xi I, \quad (11) \\ &\text{with } I = \mathbb{E}_{\boldsymbol{\mu} \sim p_\theta(\boldsymbol{\mu})} \left[\mathrm{Render}(\boldsymbol{\mu}, \phi(\mu), \pi_k)\right]. \end{aligned}$$

Here, Render$(\cdot)$ is the 3DGS rasterization [11], from camera π_k, of the set of M Gaussians with means $\{\mu_i\}_{i=0}^{M-1}$ that are sampled according to Sec. 3.2. Within the rendering operator, the Gaussian attributes $\phi(\mu_i)$ are obtained by querying attribute hash grids [18] at each location μ_i.

In all of our experiments, the expected values for an image I and image gradients $\nabla_\xi I$ in Eq. (11) are estimated using Monte Carlo estimation with a single sample from the joint distribution $p_\theta(\mu)$, *i.e.*, at every iteration we use a single set of N Gaussians sampled from the hashed probability pyramid. As is usual for optimization using Monte Carlo sampling, our goal is to find an unbiased estimator for the gradients and to reduce its variance to make optimization faster and more robust.

4.1. Gradient Estimators and Variance Reduction

The gradient in Eq. (11) contains two terms. The first term is the gradient of the loss function with respect to the image, which is trivial to compute. The second term is also trivial to compute using automatic differentiation for the Gaussian attributes ϕ. However, since θ affects the distribution used for computing the expectation, its gradient $\nabla_\theta I$ requires more care, and the variance of different unbiased estimators can significantly change the optimization behavior.

Our sampling procedure from Sec. 3.2 is differentiable, so an obvious way to estimate $\nabla_\theta I$ would be to use automatic differentiation, equivalent to the well-known *pathwise estimator*. This is unbiased, but as we show in the supplement and through ablations in Sec. 6.5, it has high variance in practice, leading to slow training and poor convergence.

Instead, we design a new unbiased gradient estimator for $\nabla_\theta I$. It is based on the standard score function estimator,

$$\nabla_\theta I = \mathbb{E}_{\mu\sim p_\theta(\mu)}\left[I \cdot \sum_{i=0}^{M-1} \nabla_\theta \log p_\theta(\mu_i)\right], \tag{12}$$

where the summation is over all Gaussians $i = 0, ..., M-1$. We use control variates to modify it to:

$$\nabla_\theta I = \mathbb{E}_{\mu\sim p_\theta(\mu)}\left[\sum_{i=0}^{M-1} (I - I_{-i}) \cdot \nabla_\theta \log p_\theta(\mu_i)\right], \tag{13}$$

where I_{-i} is the image rendered with $M-1$ Gaussians, excluding the ith one. This greatly reduces variance by weighting the gradient contribution from each sample μ_i by its individual impact on the image. Also, the estimator remains unbiased because I_{-i} and μ_i are statistically independent, and because $\mathbb{E}[\nabla_\theta \log p_\theta] = 0$ and therefore $\mathbb{E}[I_{-i} \cdot \nabla_\theta \log p_\theta(\mu_i)] = 0$. See the supplement for additional details and intuition.

At first glance, the estimator in Eq. (13) seems impossible to use in practice, because it suggests rendering M images, one for each omitted Gaussian. But surprisingly, the differences $(I - I_{-i})$ are readily computed during auto-differentiation for the gradients with respect to the Gaussian attributes ϕ. Indeed, in the supplement we prove that:

$$I - I_{-i} = o_i \frac{\partial I}{\partial o_i}, \tag{14}$$

where o_i is the opacity of the ith Gaussian. This means that our unbiased estimator can be computed using the right side of Eq. (14), providing similar computational efficiency as the gradients that would be obtained by automatic differentiation, but with much lower variance.

4.2. Loss Function

We follow 3DGS and use a combination of L_1 and D-SSIM losses. In addition, inspired by 3DGS-MCMC [12], we apply L_1 regularization to the opacities o_i and scales s_i. To encourage the model to prioritize lower order spherical harmonic coefficients, we apply a loss to the spherical harmonics coefficients $c_{i,\ell}$ of order $\ell \geq 1$. In total, the loss for the kth image is:

$$\begin{aligned}\mathcal{L} = {} & \lambda_1 \|I - I^{\mathrm{GT}}\|_1 + (1-\lambda_1)\mathrm{SSIM}(I, I^{\mathrm{GT}}) \\ & + \sum_{i\in\mathcal{X}_k} \left(\lambda_o |o_i| \mathbb{1}[o_i > \tau] + \lambda_s \|s_i\|_1 + \lambda_c \|w \odot c_i\|_1\right),\end{aligned} \tag{15}$$

where $\mathcal{X}_k$ denotes the Gaussians within the view frustum of camera π_k, $\tau = 0.05$ is a threshold for the opacity loss, and w is a weighting term for the different spherical harmonics terms, which we set to 0 for $\ell = 0$ and 0.2^ℓ for $\ell \geq 1$. We set $\lambda_1 = 0.8$, $\lambda_o = 0.05$, $\lambda_s = 0.02$, and $\lambda_c = 10^{-3}$ for all of our experiments.

5. Implementation details

5.1. Distribution Optimization

Due to the stochastic representation of the scene's geometry, we need to take extra care when sampling from the distribution and when optimizing its parameters θ.

Defensive Sampling. A well-known problem with gradient-based optimization of distributions using sampling is that if at any point a certain part of the distribution incorrectly reaches zero probability density, the model will never be able to recover. To prevent this issue, we add noise to our sampling process. We do this by adding Gaussian noise to 20% of the samples (prior to mapping them to world space using Eq. (9)). Our Gaussian noise has a standard deviation of $2 \cdot 10^{-3}$ at the beginning of training, and it is linearly annealed to zero over the first 20 thousand iterations.

We find that our model already does a good job of localizing mass without defensive sampling due to the hierarchical nature of the probability grids, but the added noise early in training helps our model explore space over more iterations.

Sample Rounding. Randomness in the exact positions of the samples could cause the distances of the Gaussians from the camera to change, which causes slight changes in the renderings and introduces variance in our estimators. In order to resolve this and lower variance, we "round" the position of every sample to lie at the center of its bin at the highest grid resolution:

$$\mu' = \frac{\lfloor \mu N_{L-1} \rfloor + 1/2}{N_{L-1}}. \tag{16}$$

Note that this does not affect our gradients since we use our gradient estimator from Sec. 4.1.

Duplicate Sample Removal. Another source of variance in the image and gradient estimators arises from the nonlinearity of the alpha compositing operation used for rendering. Because of our sample rounding, multiple samples in the same bin will overlap perfectly with one another, affecting their overall opacity. We reduce the variance caused by this overlap by removing duplicate samples, *i.e.*, after rounding the samples using Eq. (16), we discard duplicate positions and only render a unique set of Gaussian centers.

This process has another advantage: as the model keeps optimizing, it tends to place larger probabilities at parts of the scene which matter more. This causes the number of duplicate samples to increase, which means that after duplicate removal, the number of Gaussian we render decreases over time. This gives our model a natural way of pruning the number of Gaussians over training. In our experiments we start optimization with $N = 1.5 \cdot 10^7$ Gaussians, which the model lowers to an automatically-determined number. For our experiments, in order to make our comparisons fair, we set a lower threshold to the number of Gaussians equal to the number of primitives reported by 3DGS-MCMC [12]. We do this by resampling from p_θ when the number of unique primitives in our collection of sampled Gaussians drops below this threshold.

Refinement with Standard Optimization. After optimizing the probability and attribute grids, we sample the distribution and evaluate the attributes to get a collection of Gaussians, each with its own mean and set of attributes. We then refine these Gaussians by running 5K steps of standard gradient-based optimization. This slightly improves the rendering quality of our method. We report the results with and without this refinement step in our experiments.

5.2. Gaussian Attribute Grid Settings

The Gaussian attributes ϕ are represented as a multi-resolution hash encoded grid. The grids have 17 feature channels total, which are split into three parts, each of which is mapped by a small MLP to features representing opacity (using 1 channel), color (8 channels), scale and rotation (8 channels for both attributes). See supplement for a full description of the parameterization and architecture of the grid and mapping. Importantly, because of sample rounding, the value queried inside every bin of the distribution is constant, which lowers the variance of our estimators. See supplement for our initialization scheme of the attributes.

We find that when optimizing Gaussian scales, distant Gaussians tend to be small, leading to wasted capacity and a need for many Gaussians to represent the background. To resolve this, we multiply the scale parameters by the inverse square root of the Jacobian of the contraction function, as done in NeRF-Casting [22]. This factor ensures that Gaussians whose optimizable scale parameters are the same will project to approximately the same sized 2D Gaussians, regardless of their distance from the center of the scene.

5.3. Hashed Probability Pyramid Settings

We set the hashed probability pyramid to have $L = 12$ levels with the coarsest one $p_\theta^{(0)}$ having resolution $N_0 = 2$, and the finest having resolution $N_{11} = N_0 \cdot 2^{11} = 4096$. We set a budget of $B = 2^{18}$ $2 \times 2 \times 2$ distributions, meaning that each level has at most 2^{21} parameters. This means that the first levels $p_\theta^{(0)}, ..., p_\theta^{(6)}$ are dense (*i.e.*, they have no parameter sharing), whereas all other levels $p_\theta^{(7)}, ..., p_\theta^{(11)}$ contain B $2 \times 2 \times 2$ distributions which tile their respective grids.

5.4. Scene Scale Normalization

Similar to mip-NeRF 360 [1], we normalize the camera centers such that their principal components align with the world X, Y, and Z axes, and the camera centers are contained within the cube $[-1, 1]^3$. This normalization transformation is estimated using the training cameras and then applied to the testing cameras during evaluation to ensure alignment.

5.5. Additional training details

To avoid unnecessary computations and speed up training, we cull Gaussians whose centers lie outside the current camera view frustum (within a small margin). This reduces hash-grid queries to only Gaussians that can contribute to the rendered image.

We train our model with uniformly-sampled random background colors $c_{\text{bg}} \in [0, 1/2]^3$ and evaluate the model with black background $c_{\text{bg}} = 0$. We observe that these darker, randomized backgrounds help in early training: the model can use the background to approximate shadows, which temporarily frees up that capacity to allocate elsewhere without pushing opacity or probability to zero for Gaussians representing black image pixels.

6. Experiments

6.1. Datasets and Metrics

We evaluate our method on all scenes from mip-NeRF 360 [1], as well as two scenes from Tanks & Temples [13] and two scenes from Deep Blending [8]. We report standard image-based metrics for novel view synthesis: Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM) [23], and Learned Perceptual Image Patch Similarity (LPIPS) [26].

6.2. Training

Empirically, we observe that outdoor mip-NeRF 360 scenes converge faster than indoor or other dataset scenes. Therefore we train outdoor scenes for 35K total iterations (30K iterations of our probabilistic model and 5K refinement iterations, see Sec. 5.1), and indoor scenes for 65K iterations (60K probabilistic and 5K refinement steps). All experiments are conducted on a single NVIDIA H200 GPU, requiring approximately 3.5 hours for 35K iterations and 7.25 hours for 65K iterations.

6.3. Baselines

We fix the total number of training iterations across methods, following the same procedure of training outdoor mip-NeRF 360 scenes for 35K iterations and all other scenes for 65K iterations. We report results for Taming-3DGS [16] as a proxy for standard 3DGS since it allows for extended training with a fixed budget of Gaussians and for MCMC as a representative state-of-the-art baseline. Both methods are evaluated under structure-from-motion (SfM) and random initializations, with Taming-3DGS adapted to follow MCMC-3DGS's procedure of sampling 100K points uniformly within a cube bounded by the camera centers. If a baseline trained for 65K performs better at 35K than 65K, we report those values instead. Our key point of comparison is with MCMC initialized randomly (MCMC-Random), as it shares the same initialization strategy and training budget as our method, enabling a direct assessment of modeling and performance differences.

6.4. Results

We report the average of each metric for each dataset in Tab. 1 along with qualitative examples in Fig. 4, and include additional qualitative examples in the supplement. Our method achieves state-of-the-art performance among randomly initialized models, outperforming all prior approaches in overall dataset PSNR. Compared to models that are initialized using structure from motion, our approach attains competitive results, closing much of the performance gap despite starting from random initialization.

This demonstrates that our sampling-based training procedure can recover high-quality geometry and appearance purely from image supervision. Overall, these results highlight the effectiveness and robustness of our approach: it does not require SfM points for initialization, yet it achieves rendering quality comparable to methods that require them. This enables a simple, heuristic-free, end-to-end training procedure that generalizes across diverse scenes.

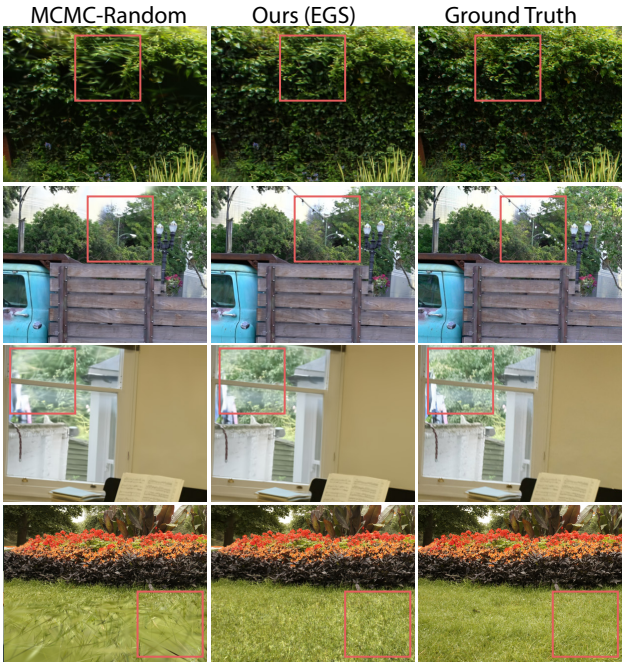

Figure 4. Qualitative comparison between MCMC-Random, our method (EGS), and ground truth. Our model more faithfully captures structural and global scene details—for example, it reconstructs the string lights (row 2) and distant trees (row 3) that MCMC-Random misses. It also captures more fine details in the leaves (row 1) and grass (row 4), with fewer visible artifacts.

6.5. Ablations

We validate our design choices and quantify their impact on the final performance of the model in Tab. 2. The most influential component is our custom gradient estimator: without it, training becomes significantly less stable and converges to much worse results.

Opacity regularization also provides a major benefit. Without it, large opaque Gaussians occlude signal beneath them and training stalls (see qualitative comparisons in the supplement). Sample rounding provides a clear advantage before refinement, but the advantage narrows after refinement. This is expected: variance reduction helps during training, but once Gaussians are represented as discrete primitives, refinement can optimize these small discrepancies directly. Finally, scaling regularization and defensive sampling each offer modest gains to model performance and additionally benefit training stability.

		COLMAP		Random			
		Taming-3DGS	**MCMC**	**Taming-3DGS**	**MCMC**	**Ours**	**Ours (refined)**
		PSNR↑/SSIM↑/LPIPS↓	PSNR↑/SSIM↑/LPIPS↓	PSNR↑/SSIM↑/LPIPS↓	PSNR↑/SSIM↑/LPIPS↓	PSNR↑/SSIM↑/LPIPS↓	PSNR↑/SSIM↑/LPIPS↓
Mip-NeRF 360 [1]	Bicycle	25.89 / 0.79 / 0.21	26.18 / 0.81 / 0.18	24.26 / 0.68 / 0.33	26.08 / 0.81 / 0.19	26.12 / 0.80 / 0.23	26.31 / 0.80 / 0.22
	Garden	28.19 / 0.88 / 0.11	28.23 / 0.88 / 0.10	26.60 / 0.83 / 0.16	27.97 / 0.88 / 0.11	28.02 / 0.88 / 0.13	28.34 / 0.88 / 0.12
	Stump	26.88 / 0.78 / 0.23	27.48 / 0.82 / 0.19	19.29 / 0.46 / 0.46	27.39 / 0.81 / 0.20	27.22 / 0.81 / 0.22	27.35 / 0.81 / 0.21
	Flowers	22.17 / 0.63 / 0.35	22.29 / 0.66 / 0.27	20.79 / 0.57 / 0.37	22.21 / 0.66 / 0.27	21.97 / 0.65 / 0.32	22.22 / 0.65 / 0.31
	Treehill	23.21 / 0.66 / 0.35	23.20 / 0.67 / 0.27	22.18 / 0.62 / 0.40	23.17 / 0.67 / 0.27	23.06 / 0.67 / 0.34	23.23 / 0.67 / 0.33
	Kitchen	32.42 / 0.94 / 0.13	32.54 / 0.94 / 0.13	31.23 / 0.93 / 0.15	32.45 / 0.94 / 0.14	32.19 / 0.93 / 0.15	32.48 / 0.93 / 0.15
	Counter	29.84 / 0.92 / 0.23	29.80 / 0.93 / 0.21	28.86 / 0.89 / 0.28	29.60 / 0.92 / 0.22	29.47 / 0.92 / 0.23	29.65 / 0.92 / 0.23
	Bonsai	33.44 / 0.95 / 0.22	33.12 / 0.96 / 0.21	31.92 / 0.93 / 0.24	32.94 / 0.95 / 0.21	32.75 / 0.95 / 0.23	33.15 / 0.95 / 0.22
	Room	32.70 / 0.93 / 0.25	32.58 / 0.94 / 0.23	31.04 / 0.91 / 0.30	32.50 / 0.94 / 0.24	32.09 / 0.93 / 0.26	32.41 / 0.93 / 0.25
	Average	28.30 / 0.83 / 0.23	28.38 / 0.84 / 0.20	26.24 / 0.76 / 0.30	28.26 / 0.84 / 0.21	28.10 / 0.84 / 0.23	28.35 / 0.84 / 0.23
	Avg Outdoor	25.27 / 0.75 / 0.25	25.48 / 0.77 / 0.20	22.62 / 0.63 / 0.34	25.36 / 0.77 / 0.21	25.28 / 0.76 / 0.25	25.49 / 0.76 / 0.24
	Avg Indoor	32.1 / 0.93 / 0.21	32.01 / 0.94 / 0.20	30.76 / 0.91 / 0.24	31.87 / 0.94 / 0.20	31.62 / 0.93 / 0.22	31.92 / 0.93 / 0.21
T&T [13]	Train	22.94 / 0.83 / 0.23	22.94 / 0.84 / 0.21	21.96 / 0.78 / 0.28	22.25 / 0.82 / 0.24	22.50 / 0.81 / 0.24	22.53 / 0.81 / 0.24
	Truck	26.12 / 0.89 / 0.15	26.38 / 0.90 / 0.13	23.56 / 0.84 / 0.21	26.10 / 0.89 / 0.13	26.20 / 0.88 / 0.17	26.28 / 0.89 / 0.17
	Average	24.53 / 0.86 / 0.19	24.66 / 0.87 / 0.17	22.76 / 0.81 / 0.25	24.17 / 0.85 / 0.19	24.35 / 0.84 / 0.20	24.40 / 0.85 / 0.20
DB [8]	Dr Johnson	29.49 / 0.90 / 0.31	29.43 / 0.90 / 0.30	28.84 / 0.89 / 0.34	29.12 / 0.89 / 0.31	28.98 / 0.90 / 0.32	29.21 / 0.90 / 0.31
	Playroom	29.76 / 0.90 / 0.30	29.78 / 0.90 / 0.30	29.69 / 0.90 / 0.31	30.32 / 0.91 / 0.29	30.25 / 0.91 / 0.30	30.49 / 0.91 / 0.30
	Average	29.62 / 0.90 / 0.31	29.60 / 0.90 / 0.30	29.26 / 0.90 / 0.33	29.72 / 0.90 / 0.30	29.61 / 0.91 / 0.31	29.85 / 0.91 / 0.30

Table 1. Reconstruction metrics of our method compared with Taming-3DGS [16] and 3DGS-MCMC [12], on the mip-NeRF 360 [1], Tanks & Temples (T&T) [13], and Deep Blending (DB) [8] datasets. We show the results of methods using random initialization on the right, and methods initialized using COLMAP points on the left. We include results for our model before the refinement procedure described in Sec. 5.1 (second to last column) as well as after 5K steps of refinement (last column).

Configuration	**Pre-refinement**	**Post-refinement**
	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓
A) No opacity regularization	26.32 / 0.77 / 0.31	26.74 / 0.78 / 0.30
B) No scaling regularization	29.26 / 0.88 / 0.20	29.48 / 0.88 / 0.19
C) No sample rounding	28.85 / 0.87 / 0.22	29.44 / 0.88 / 0.20
D) No defensive sampling	29.20 / 0.88 / 0.20	29.45 / 0.88 / 0.19
E) No custom gradient (autodiff)	22.35 / 0.62 / 0.51	23.69 / 0.64 / 0.47
F) Ours (full model)	29.29 / 0.88 / 0.20	29.52 / 0.88 / 0.19

Table 2. We ablate our model on six scenes (three outdoor scenes and three indoor scenes) from mip-NeRF 360. We measure the effects of opacity and scaling regularization (A and B), sample rounding (C), defensive sampling (D) and our custom gradient estimator (E). We also quantify the effects of refinement by reporting all values before and after 5K refinement iterations.

7. Discussion & Future Work

Eulerian Gaussian Splatting (EGS) introduces a probabilistic framework for fitting a collection of Gaussians that faithfully represent a set of multiview images. Unlike prior 3DGS-based approaches, our method relies solely on gradient cues and requires no heuristics to prevent convergence to poor local minima.

We introduce Hashed Probability Pyramids, a compact yet expressive representation for high-resolution continuous probability distributions, and a control variate gradient estimator that stabilizes training despite the inherent variance of our probabilistic formulation. Together, these components enable reliable and heuristic-free optimization.

While our approach marks a significant step toward more robust and principled optimization, it remains slower and more memory-intensive than standard 3DGS due to the large number of samples and hash-grid queries required during training. Future work could explore more efficient sampling and variance reduction strategies to further improve training speed and memory efficiency, bringing probabilistic splatting closer to real-time practicality.

In summary, EGS offers a principled bridge between continuous-field optimization and discrete Gaussian rendering, demonstrating that sampling-based formulations can achieve high-quality reconstructions without heuristic intervention. By using an optimizable probabilistic model for 3D Gaussians, our framework combines the efficiency of standard splatting with the robustness of continuous fields.

Acknowledgements. This work was supported in part by the NVIDIA Academic Grant Program and NSF cooperative agreement PHY2019786 (an NSF AI Institute, iaifi.org).

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022. 2, 4, 6, 7, 8

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2023. 2

[3] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising Densification in Gaussian Splatting, 2024. arXiv preprint. 2

[4] Xiaobin Deng, Changyu Diao, Min Li, Ruohan Yu, and Duanqing Xu. Efficient Density Control for 3D Gaussian Splatting, 2025. arXiv preprint. 2

[5] Shrisudhan Govindarajan, Zeno Sambugaro, Ahan Shabhanov, Towaki Takikawa, Daniel Sun, Weiweiand Rebain, Nicola Conci, Kwang Moo Yi, and Andrea Tagliasacchi. Lagrangian hashing for compressed neural field representations. In *ECCV*, 2024. 2

[6] Shrisudhan Govindarajan, Daniel Rebain, Kwang Moo Yi, and Andrea Tagliasacchi. Radiant foam: Real-time differentiable ray tracing. *arXiv preprint arXiv:2502.01157*, 2025. 2

[7] Florian Hahlbohm, Linus Franke, Moritz Kappel, Susana Castillo, Martin Eisemann, Marc Stamminger, and Marcus Magnor. INPC: Implicit Neural Point Clouds for Radiance Field Rendering, 2025. arXiv preprint. 2

[8] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. 37(6):257:1–257:15, 2018. 7, 8

[9] Jan Held, Renaud Vandeghen, Adrien Deliege, Abdullah Hamdi, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. *arXiv*, 2025. 2

[10] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. Relu fields: The little non-linearity that could. In *ACM SIGGRAPH 2022 conference proceedings*, pages 1–9, 2022. 2

[11] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering, 2023. SIGGRAPH 2023. 1, 2, 5

[12] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3D Gaussian Splatting as Markov Chain Monte Carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. Spotlight Presentation. 1, 2, 5, 6, 8

[13] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 7, 8

[14] Rong Liu, Dylan Sun, Meida Chen, Yue Wang, and Andrew Feng. Deformable beta splatting, 2025. 2

[15] Alexander Mai, Peter Hedman, George Kopanas, Dor Verbin, David Futschik, Qiangeng Xu, Falko Kuester, Jonathan T Barron, and Yinda Zhang. EVER: Exact volumetric ellipsoid rendering for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4930–4939, 2025. 2

[16] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3dgs: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery. 2, 7, 8

[17] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Eur. Conf. Comput. Vis.*, 2020. 1, 2

[18] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. SIGGRAPH 2022. 3, 5

[19] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 2, 4

[20] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. MIT Press, 4th edition, 2023. 3

[21] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5459–5469, 2022. 2

[22] Dor Verbin, Pratul P Srinivasan, Peter Hedman, Ben Mildenhall, Benjamin Attal, Richard Szeliski, and Jonathan T Barron. NeRF-Casting: Improved view-dependent appearance with consistent reflections. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–10, 2024. 6

[23] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 7

[24] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2(3):6, 2021. 2

[25] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5752–5761, 2021. 2

[26] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–595, 2018. 7