Supplementary Material

Eulerian Gaussian Splatting using Hashed Probability Pyramids

Table of Contents

S1 Hashed Probability Pyramids **1**
S1.1 Hashing Function . . . 1
S1.2 Probability Grid Normalization . . . 1
S1.3 Additional Remarks . . . 2

S2 Gradient Estimator **2**
S2.1 Estimator Intuition . . . 2
S2.2 Pathwise Estimator . . . 4
S2.3 Empirical Evidence for Variance Reduction . . . 4
S2.4 Gradient Estimator Derivation . . . 5

S3 Implementation Details **6**
S3.1 Hash Encoding Grid Settings. . . . 6
S3.2 Custom Gradient Implementation . . . 7

S4 Additional Training Details and Results **7**
S4.1 Renderer modifications . . . 7
S4.2 Refinement details . . . 8
S4.3 Near Plane Culling . . . 8
S4.4 Additional Qualitative Comparisons . . . 8
S4.5 Initialization with COLMAP . . . 9
S4.6 Dependence on the number of samples . . . 9

S1 Hashed Probability Pyramids

S1.1 Hashing Function

We use the same hash encoding function as Instant-NGP [7] to map samples from a bin at level ℓ to a $2 \times 2 \times 2$ block of level $\ell + 1$. For a level ℓ with resolution N_ℓ satisfying $N_\ell^3 \leq 8B$, the mapping T_ℓ is the identity mapping, meaning that the bin index ν gets mapped to the $2 \times 2 \times 2$ distribution at the index ν. For levels ℓ with $N_\ell^3 > 8B$, we set:

$$T(i, j, k) = (i \cdot \pi_1 \oplus j \cdot \pi_2 \oplus k \cdot \pi_3) \bmod B, \tag{1}$$

where $\nu = (i, j, k)$ is the 3D index, π_1, π_2, π_3 are three large prime numbers (chosen to be the same as [7]), and $\oplus$ denotes the bitwise XOR operation. Note that even though all levels share the same π_1, π_2, π_3 the geometric scaling of the grid resolution N_ℓ effectively decorrelates the hash collisions across the hierarchy. This is because the same i, j, k indices correspond to radically different spatial coordinates in different levels.

A hashed probability pyramid with the experimental settings described in Sec. 5.3 uses 86 million parameters to represent a 4096^3 grid. This requires 0.33 GB of memory, which is smaller than the memory required to store a dense 4096^3 grid (275 GB) by a factor of more than 800.

S1.2 Probability Grid Normalization

In order to sample from our hashed probability pyramid we need to normalize all per-level distributions. More specifically, we need a procedure for creating grids of (normalized) distributions of resolution $R \times R \times R$. For the coarsest distribution $\ell = 0$, we have $R = N_0$, and for all other levels $\ell > 1, ..., L - 1$, $R = 2$. Given a grid of $R \times R \times R$ parameters θ_ν where $\nu \in \{0, ..., R - 1\}^3$, we do this by defining:

$$p(\mu) = R^3 \cdot \frac{\sum_\nu \exp(\theta_\nu) \mathbb{1}\left[\lfloor \mu R \rfloor = \nu\right]}{\sum_{\nu'} \exp(\theta_{\nu'})}, \tag{2}$$

for all $\mu \in [0,1]^3$. The factor R^3 is needed to satisfy $\int p(\mu)d\mu = 1$. Here, each $p(\mu)$ is an $R \times R \times R$ block of $f_\theta^{(\ell)}(\mu)$.

Note that this normalization operation must be computed at most B times per level. This is because for levels ℓ with parameter sharing (*i.e.*, levels ℓ such that $N_\ell^3 > 8B$) there are only B distributions, and for levels without parameter sharing there are fewer than B distributions.

S1.3 Additional Remarks

In addition to making high-resolution probability grids tractable, hashed probability pyramids impose a natural coarse-to-fine hierarchy that aids learning. Spatially nearby samples share coarse-level blocks before diverging at finer resolutions, and this promotes large-scale smoothness while preserving local detail. Importantly, the model automatically automatically allocates capacity across scales through end-to-end optimization, without manually-designed heuristics or human intervention.

Another notable property of our sampling-based approach is that the gradient updates naturally drive down the probabilities of Gaussians that do not contribute to any of the rendered images. Specifically, since the hashed probability pyramid is normalized globally to integrate to one, each time the model drives the probability up at a bin, it also reduces the probabilities everywhere else. This includes reducing the probability in bins with Gaussian samples that are occluded or outside any of the camera views, never contributing to the gradient directly. This in essence acts as an implicit regularizer during training.

S2 Gradient Estimator

S2.1 Estimator Intuition

In Sec. 4.1 of the main paper, we describe various gradient estimators, and in Sec. 6.5 we show through ablations that our unbiased control variate gradient estimator outperforms the pathwise estimator provided by autodiff. Here we elaborate on these findings. We first provide mathematical intuition using a simplified 1D additive rendering model. Then, in Sec. S2.3, we further support these conclusions with an empirical analysis of the gradients produced by each method for the full 3D problem.

Our simplified 1D rendering model replaces alpha compositing with summation. We write it as

$$F(\mu) = \sum_{i=0}^{M-1} f(\mu_i), \tag{3}$$

where $f(\cdot)$ is the rendering of the ith Gaussian whose center is μ_i.

As in the main paper, our rendering model is probabilistic:

$$I = \mathbb{E}_{\mu \sim p_\theta(\mu)} \left[F(\mu) \right], \tag{4}$$

where $p_\theta(\mu)$ is the joint probability distribution over a collection of M Gaussian centers in one dimension, $\mu = (\mu_0, ..., \mu_{M-1}), \mu_i \in \mathbb{R}$.

Our goal is to obtain an unbiased estimator for the gradient of this rendering model $\nabla_\theta I$ with variance that is as low as possible. One common unbiased estimator used in machine learning and computer graphics is the score-based estimator:

$$\begin{aligned} g_j &\triangleq \nabla_\theta \mathbb{E}_{\mu \sim p_\theta(\mu)} \left[F(\mu) \right] \\ &= \nabla_\theta \int p_\theta(\mu) F(\mu) \, d\mu \\ &= \int p_\theta(\mu) F(\mu) \nabla_\theta \log p_\theta(\mu) \, d\mu \\ &= \mathbb{E}_{\mu \sim p_\theta(\mu)} \left[\sum_{i=0}^{M-1} F(\mu) \nabla_\theta \log p_\theta(\mu_i) \right]. \end{aligned} \tag{5}$$

Here, we used the independence of samples, $\log p_\theta(\mu) = \sum_{i=0}^{M-1} \log p_\theta(\mu_i)$.

Alternatively, we can use the linearity of our additive rendering model to derive a score-based estimator using the marginal

distribution over a single Gaussian instead of the joint M-Gaussian distribution:

$$
\begin{aligned}
g_m &\triangleq \nabla_\theta \mathbb{E}_{\mu \sim p_\theta(\mu)} \left[\sum_{i=0}^{M-1} f(\mu_i) \right] \\
&= \sum_{i=0}^{M-1} \nabla_\theta \mathbb{E}_{\mu_i \sim p_\theta(\mu)} [f(\mu_i)] \\
&= \sum_{i=0}^{M-1} \mathbb{E}_{\mu_i \sim p_\theta(\mu)} [f(\mu_i) \nabla_\theta \log p_\theta(\mu_i)].
\end{aligned} \tag{6}
$$

The two estimators in Eq. (5) and Eq. (6) look similar, but their variances are extremely different. The marginal distribution estimator in Eq. (6) provides significantly lower variance than the estimator in Eq. (5). To show this analytically, we begin by expanding the expression of the joint estimator:

$$
\begin{aligned}
g_j &= \nabla_\theta \mathbb{E}_{\mu \sim p_\theta(\mu)} [F(\mu)] \\
&= \mathbb{E}_{\mu \sim p_\theta(\mu)} \left[\sum_{i=0}^{M-1} f(\mu_i) \cdot \sum_{j=0}^{M-1} \nabla_\theta \log p_\theta(\mu_j) \right] \\
&= \mathbb{E}_{\mu \sim p_\theta(\mu)} \left[\sum_{i=0}^{M-1} f(\mu_i) \cdot \nabla_\theta \log p_\theta(\mu_i) + \sum_{i=0}^{M-1} \nabla_\theta \log p(\mu_i) \sum_{j \neq i} f(\mu_j) \right] \\
&= \underbrace{\mathbb{E}_{\mu \sim p_\theta(\mu)} \left[\sum_{i=0}^{M-1} f(\mu_i) \cdot \nabla_\theta \log p(\mu_i) \right]}_{g_m} + \underbrace{\mathbb{E}_{\mu \sim p_\theta(\mu)} \left[\sum_{i=0}^{M-1} \nabla_\theta \log p(\mu_i) \sum_{j \neq i} f(\mu_j) \right]}_{R}.
\end{aligned} \tag{7}
$$

Observe that the first term in Eq. (7) is identical to the marginal estimator g_m in Eq. (6). The additional "cross-term" R has zero mean, and we will show that it adds variance to the entire estimator. Let $s_i = \nabla_\theta \log p_\theta(\mu_i)$ and $a_j = f(\mu_j)$, and define per-sample moments:

$$
\mathbb{E}[a_i] = \mu_a, \quad \mathrm{Var}(a_i) = \sigma_a^2, \quad \mathbb{E}[s_i] = 0, \quad \mathrm{Var}(s_i) = \sigma_s^2, \quad \rho = \mathbb{E}[a_i s_i]. \tag{8}
$$

The cross-term R is $\sum_i \sum_{j \neq i} s_i a_j$ and we can write its variance as:

$$
\begin{aligned}
\mathrm{Var}(R) &= \mathbb{E}[R^2] - (\mathbb{E}[R])^2 \\
&= \mathbb{E}\left[\left(\sum_i \sum_{j \neq i} s_i a_j \right)^2 \right] - 0 \\
&= \sum_i \sum_{j \neq i} \sum_k \sum_{l \neq k} \mathbb{E}[s_i a_j s_k a_l] \\
&= \sigma_s^2 \left(M(M-1)\sigma_a^2 + M(M-1)^2 \mu_a^2 \right) + M(M-1)\rho^2.
\end{aligned} \tag{9}
$$

Notably, the variance of the cross-term is $\mathcal{O}(M^3)$. Likewise, we can derive the variance of the marginal estimator:

$$
\begin{aligned}
\mathrm{Var}(g_m) &= \sum_{i=0}^{M-1} \mathrm{Var}(f(\mu_i) \nabla_\theta \log p_\theta(\mu_i)). \\
&= M \cdot \mathrm{Var}(a_i s_i),
\end{aligned} \tag{10}
$$

where we used the independence μ_i and μ_j for $i \neq j$.

This means that the variance of the marginal estimator is $\mathcal{O}(M)$. Because $|\operatorname{Cov}(g_m, R)| \leq (\mathrm{Var}(g_m)\,\mathrm{Var(R)})^{1/2}$, $\operatorname{Cov}(g_m, R)$ is at most $\mathcal{O}(M^2)$. This means that the variance of R, which is $\mathcal{O}(M^3)$, dominates the variance for sufficiently large M since $\mathrm{Var}(g_j) = \mathrm{Var}(g_m + R) = \mathrm{Var}(g_m) + \mathrm{Var}(R) + 2\operatorname{Cov}(g_m, R)$.

The implication is that the variance of the joint M-Gaussian gradient estimator scales with the number of Gaussians M much more quickly than the marginal estimator. In contrast, the marginal estimator negates these cross-terms so that updates to a voxel of the probability distribution are only influenced by Gaussians generated by that voxel, which leads to more stable and lower-variance training.

Our estimator in Sec. 4.1 of the main paper is inspired by this exact analysis. In this simplified 1D additive problem, the intuitive reason that the joint estimator of Eq. (5) results in much higher variance is that the gradient update of the ith sample stemming from $\log p_\theta(\mu_i)$ is multiplied by the entire image $F(\mu)$ instead of just the contribution of the ith Gaussian to the image $f(\mu_i)$. This "mixing" of contributions across all Gaussians adds variance to the estimator. In the full 3D problem with alpha compositing, there is no analytical equivalent to the marginal estimator, but we can leverage the same intuition by replacing the full image I in the gradient expression with $(I - I_{-i})$, the contribution of the ith Gaussian.

S2.2 Pathwise Estimator

As mentioned in the main paper, because our sampling process in Sec. 3.2.1 is differentiable, we can derive a third gradient estimator—the pathwise estimator—using the "reparameterization trick". If we denote our sampling process by $\mu_i = g(u_i, \theta)$, which maps $u_i \sim \mathrm{U}([0,1])$ to $\mu_i \sim p_\theta$, we can write the pathwise estimator as

$$\nabla_\theta F(\mu) = \mathbb{E}_{u\sim \mathrm{U}([0,1])}\left[\sum_{i=0}^{M-1} \nabla_{\mu_i} F(\mu) \nabla_\theta g(u_i, \theta)\right]. \tag{11}$$

This is another unbiased estimator, and it only requires backpropagation through the renderer to evaluate $\nabla_{\mu_i} F(\mu)$.

In the next section, we include a comparison of the variances of the three estimators for our 1D problem, showing that the marginal score-based estimator has the lowest variance. We then show that the analogous estimator of the original 3D problem, which we introduced in Sec. 4.1 of the original paper, has lower variance than the alternative estimators in 3D empirical studies.

S2.3 Empirical Evidence for Variance Reduction

We showed through ablations in Sec. 6.5 of the main paper that our model trains much better with our control variate estimator than with standard auto differentiation gradients (*i.e.*, the pathwise estimator). Here we provide additional qualitative comparisons that further support this finding.

In 1D, we construct a ground-truth signal by summing 15 Gaussians with randomly sampled widths, amplitudes, and positions, as shown on the left of Fig. S1. We initialize a uniform probability distribution representing the likelihood of placing a Gaussian center at each position along the signal. The middle plot shows the gradient of this probability distribution. The rightmost plot compares gradient variance across the three estimators: the marginal estimator, the joint M-Gaussian estimator, and the pathwise estimator. Among these, the marginal estimator exhibits the lowest overall variance (particularly in empty regions of the signal) indicating that probabilities in these areas decrease reliably. In contrast, the joint M-Gaussian estimator has the highest variance.

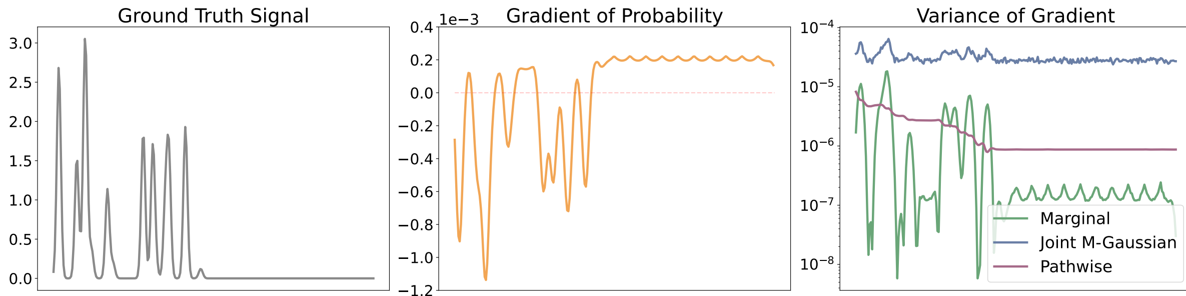

Figure S1. 1D illustration of the three gradient estimators. The ground-truth signal (left) is constructed by summing 15 Gaussians with random widths, amplitudes, and positions. We initialize a uniform probability distribution over possible Gaussian center locations and visualize its gradient (middle). The gradient variance (right) is compared across three estimators: marginal, joint M-Gaussian, and pathwise. The marginal estimator exhibits the lowest overall variance, while the joint M-Gaussian estimator has the highest.

In 3D, we visualize the estimated gradients of our probability distribution for three estimators—pathwise, standard score function, and our proposed control variate, which is inspired by the marginal estimator from the 1D experiments. We define a

single level, dense 128^3 probability grid (*i.e.*, $L = 1$, $N_0 = 128$, and $B > 2^{18}$) initialized uniformly. Using a single image from the bicycle scene in the mip-NeRF 360 dataset [2], downsampled by a factor of 16 (4 times smaller than standard training resolution), we compute 100 independent gradient estimates of the loss for each method, with each estimate using one million samples. We then visualize the mean and variance across these 100 gradient estimates.

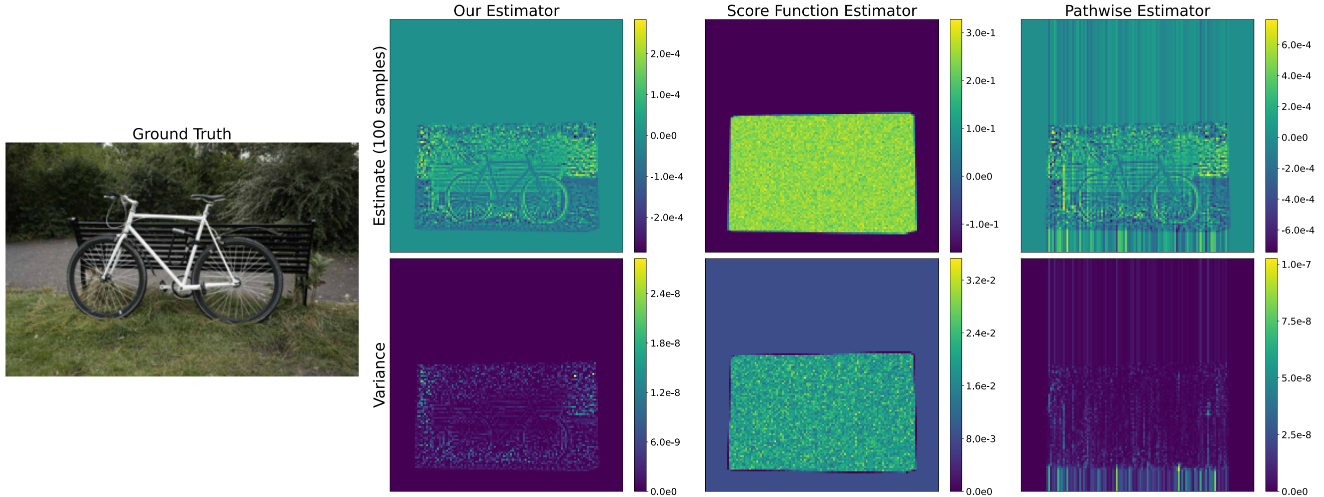

Figure S2. 3D gradient variance comparison across estimators. We visualize the mean (top) and variance (bottom) of 100 independently estimated probability gradients for the bicycle scene from mip-NeRF 360 [2]. Our control variate estimator exhibits substantially lower variance than the pathwise and standard score function estimators, particularly in empty regions of space, leading to more stable and efficient training.

Each column of Fig. S2 shows the estimated gradient (top) and its variance (bottom) for one of the three estimators. Our control variate estimator achieves several orders of magnitude lower variance than both the pathwise and score function estimators. Visually, it maintains low variance even in regions corresponding to Gaussians outside the image plane, similar to the behavior of the marginal estimator in 1D. This is a desirable property that allows the model to reliably reduce probability in empty regions and avoid wasted capacity. In contrast, the pathwise estimator exhibits structured, striation-like variance patterns that we find detrimental to training stability in practice.

S2.4 Gradient Estimator Derivation

A key to making our gradient estimator tractable is that the difference $(I - I_{-i})$ between the full rendering of an image I and its rendering I_{-i} with the ith Gaussian removed simplifies to known quantities that are already being computed during the backward pass to update the attribute grids. The derivation follows.

The color of a pixel is determined by the Gaussians that intersect the ray that backprojects from that pixel. Let these K Gaussians have colors $c_1, ..., c_K \in \mathbb{R}^3$ and opacities $o_1, ..., o_K \in [0, 1]$. Then the value of the pixel is

$$I = \sum_{k=1}^{K} c_k \alpha_k T_k, \tag{12}$$

where the transmittance is defined as

$$T_k = \prod_{k'=1}^{k-1} (1 - \alpha_{k'}), \tag{13}$$

and α_k is the pixel opacity determined by o_k and the pixel position.

We split the color from Eq. (12) into three terms, given any Gaussian index $i \in \{1, ..., K\}$:

$$I = \sum_{k=1}^{i-1} c_k \alpha_k T_k + c_i \alpha_i T_i + \sum_{k=i+1}^{K} c_k \alpha_k T_k, \tag{14}$$

and we observe that the color obtained by omitting the ith Gaussian is

$$I_{-i} = \sum_{k=1}^{i-1} c_k \alpha_k T_k + \frac{1}{1-\alpha_i} \sum_{k=i+1}^{K} c_k \alpha_k T_k, \tag{15}$$

where the division by $1-\alpha_i$ removes the effect of the ith Gaussian from the transmittance of Gaussians with indices $k > i$. Subtracting Eq. (15) from Eq. (14) yields

$$I - I_{-i} = c_i \alpha_i T_i - \frac{\alpha_i}{1-\alpha_i} \sum_{k=i+1}^{K} c_k \alpha_k T_k. \tag{16}$$

On the other hand, taking the derivative of I from Eq. (14) with respect to α_i, we obtain

$$\frac{\partial I}{\partial \alpha_i} = c_i T_i + \frac{1}{1-\alpha_i} \sum_{k=i+1}^{K} c_k \alpha_k T_k, \tag{17}$$

which when combined with Eq. (16) yields

$$I - I_{-i} = \alpha_i \frac{\partial I}{\partial \alpha_i}. \tag{18}$$

Additionally, in the 3DGS render operation, the pixel opacity α_i is proportional to the Gaussian opacity o_i, so

$$\alpha_i \frac{\partial I}{\partial \alpha_i} = o_i \frac{\partial I}{\partial o_i}, \tag{19}$$

and we can write the difference as

$$I - I_{-i} = o_i \frac{\partial I(x)}{\partial o_i}. \tag{20}$$

Thus, computing the difference $(I - I_{-i})$ does not require an additional forward pass to render I_{-i}. Instead, it can be computed directly from the opacity o_i and its gradient $\partial \mathcal{L} / \partial o_i$, both of which are readily available from the standard rendering process. By substituting Eq. (20) into Eq. (13) from the main paper, we obtain the final expression for our gradient estimator:

$$\nabla_\theta I = \mathbb{E}_{\mu \sim p_\theta(\mu)} \left[\sum_{i=0}^{M-1} o_i \frac{\partial I}{\partial o_i} \cdot \nabla_\theta \log p_\theta(\mu_i) \right]. \tag{21}$$

S3 Implementation Details

S3.1 Hash Encoding Grid Settings.

We use small MLPs with a single hidden layer of size 32 to map the opacity and scale/rotation gridded features to output attributes, and we map color features with a single linear layer. Fig. S3 shows the exact hash encoding configurations.

Our parameterization of scale $s \in \mathbb{R}^3$ and opacity $o \in [0, 1]$ is defined as:

$$s = \text{softplus}(\tilde{s} + \text{softplus}^{-1}(s_0)), \tag{22}$$

$$o = \text{sigmoid}(\tilde{o} + \text{sigmoid}^{-1}(o_0)), \tag{23}$$

where $\tilde{o}$ and $\tilde{s}$ are the outputs of the hash encoding, and o_0 and s_o are the initial values for opacity and scale. This relies on the hash encoding output being small at the start of training, which in practice is $\sim 10^{-6}$. We initialize our model with $o_0 = 0.05$ and $s_0 = 0.0006$.

We represent the three color channels as degree three spherical harmonics. To prevent higher-order coefficients from dominating during early training, we multiply the color coefficients by a factor that depends on their degree ℓ, such that the color is:

$$C(\boldsymbol{\omega}) = \sum_{\ell=0}^{3} \sum_{m=-\ell}^{\ell} \alpha^\ell \cdot c_\ell^m Y_\ell^m(\boldsymbol{\omega}), \tag{24}$$

where $\boldsymbol{\omega}$ is the view direction and $\alpha = 0.2$, and c_ℓ^m are the optimizable grid parameters.

```
encoding_config = {
    'otype': 'HashGrid',
    'n_levels': 13,
    'n_features_per_level': 1 if opacity else 8,
    'log2_hashmap_size': 23,
    'base_resolution': 2,
    'per_level_scale': 2.0,
    'interpolation': 'Smoothstep'
}
opacity_network_config = {
    'otype': 'FullyFusedMLP',
    'activation': 'LeakyReLU',
    'output_activation': 'None',
    'n_neurons': 32,
    'n_hidden_layers': 1
}
color_network_config = {
    'otype': 'FullyFusedMLP',
    'output_activation': 'None',
    'n_hidden_layers': 0
}
scaling&rotation_network_config = {
    'otype': 'FullyFusedMLP',
    'activation': 'LeakyReLU',
    'output_activation': 'None',
    'n_neurons': 32,
    'n_hidden_layers': 1
}
```

Figure S3. Hash grid configuration for the `tiny-cuda-nn` implementation [6]

S3.2 Custom Gradient Implementation

The pathwise (auto-differentiation) gradient of the loss can be written as the sum over Gaussian centers μ_i:

$$\nabla_\theta \mathcal{L}(I, I^{\mathrm{GT}}) = \sum_{i=1}^{M-1} \frac{\partial \mathcal{L}}{\partial \mu_i} \cdot \frac{\partial \mu_i}{\partial \theta}. \tag{25}$$

By manipulating the gradients of these terms, we can force the model to use our control variate gradient estimator during the backward pass. We replace the sampled positions μ_i with a surrogate variable whose gradient tracks the log-probability:

$$\mu_i \quad \longrightarrow \quad \not\nabla(\mu_i) + \log p_\theta(\mu_i) - \not\nabla\big(\log p_\theta(\mu_i)\big), \tag{26}$$

where $\not\nabla(\cdot)$ denotes the stop-gradient operator. This ensures that gradients flow through $\log p_\theta(\mu_i)$ while the value of μ_i is preserved in the forward pass.

We also modify the upstream gradient $\partial \mathcal{L} / \partial \mu_i$, by substituting

$$\frac{\partial \mathcal{L}}{\partial \mu_i} \quad \longrightarrow \quad o_i \, \frac{\partial \mathcal{L}}{\partial o_i} \tag{27}$$

inside the renderer backward pass.

We implement these changes in practice by augmenting $\boldsymbol{\mu}$ and $\partial I(x; \boldsymbol{\mu}) / \partial \boldsymbol{\mu}$ with an additional channel to store the surrogate gradients and zero out the gradients elsewhere. These changes enable us to use our custom gradient estimator while preserving the efficiency of the standard rendering pipeline.

S4 Additional Training Details and Results

S4.1 Renderer modifications

Our rendering pipeline utilizes an adapted version of the Slang.D-based 3DGS rasterizer [1, 5], tailored to accommodate our custom gradient estimator.

Additionally, we cap the number of rendered Gaussians at 7.5 million per frame by randomly selecting 7.5 million in-frustum Gaussians to keep if more than 7.5 million Gaussian samples are within the view frustum.

S4.2 Refinement details

During the 5K refinement steps that we apply to the set of Gaussians sampled from our model, we deviate from standard optimization in a few ways. First, we continue frustum-culling the Gaussians during refinement in order to remain consistent with the sampling model. Second, we fix the positions of all Gaussians and we use a reduced learning rate of $5 \cdot 10^{-3}$ for opacity. We set all other learning rates to be consistent with the learning rates used by 3DGS at the start of training [3]. Finally, we disable all adaptive density control modules (pruning, cloning, culling) so that the set of Gaussians and their positions are fixed.

S4.3 Near Plane Culling

The standard 3DGS renderer employs a z-threshold to discard Gaussians within a small margin of the camera's image plane, mitigating artifacts caused by "floater" artifacts. We retain the default $z = 0.2$ for all datasets except Tanks and Temples [4], where we lower it to $z = 0.05$ to account for objects being positioned closer to the camera.

S4.4 Additional Qualitative Comparisons

Here we provide a few complementary visualizations for our ablation studies in Sec. 6.5. Fig. S4 shows a comparison of our full model with a variant trained without opacity regularization. The absence of opacity regularization leads to regions dominated by large, high-opacity Gaussians, resulting in poor reconstruction of fine textures such as distant grass. In contrast, opacity regularization enables better preservation of detail. Fig. S5 shows a comparison of our approach with a model trained using the pathwise gradient estimator. Although both estimators are unbiased, our proposed estimator more effectively optimizes the underlying probability distribution, yielding a more accurate scene reconstruction.

Figure S4. The results of our method with the opacity regularization (left) and without it (right). Opacity regularization prevents large high-opacity Gaussians from dominating regions such as distant grass, and enables preserves fine detail.

Figure S5. Our control variate estimator (left) produces sharper, more accurate reconstructions than the pathwise gradient estimator (right), which fails to optimize the probability distribution effectively.

S4.5 Initialization with COLMAP

We also explored initializing the weights of the hashed probability pyramid using the points output from COLMAP. Specifically, we set the probability value of occupied voxels to be twice that of empty ones. This initialization reduces the need for the model to "discover" these regions during optimization. Empirically, we find that this leads to faster convergence. We evaluate this strategy on four randomly selected scenes from mip-NeRF 360 (two indoor and two outdoor scenes) and observe that COLMAP-based initialization achieves the same PSNR approximately 15% faster compared to uniform initialization.

S4.6 Dependence on the number of samples

Our experiments indicate that increasing the number of Gaussian samples improves reconstruction quality, at the cost of moderately increased training time. We provide a quantitative comparison in Tab. 1, where we vary the number of samples on the mip-NeRF 360 "Room" scene.

As shown in the table, using more samples consistently improves PSNR and LPIPS, both before and after refinement, while SSIM remains largely stable. However, these gains come with a gradual increase in training time. This highlights a trade-off between reconstruction fidelity and computational cost, where larger sample counts yield better performance but incur higher training overhead.

# Samples	Pre-refinement PSNR↑ / SSIM↑ / LPIPS↓	Post-refinement PSNR↑ / SSIM↑ / LPIPS↓	Training Time (in hours)
A) $1.5 \cdot 10^7$	32.09 / 0.93 / 0.26	32.41 / 0.93 / 0.25	5.87
B) 10^7	32.04 / 0.93 / 0.27	32.38 / 0.93 / 0.26	5.22
C) $5 \cdot 10^6$	31.71 / 0.93 / 0.28	31.90 / 0.93 / 0.27	4.70
D) $2.5 \cdot 10^6$	31.51 / 0.92 / 0.29	31.70 / 0.92 / 0.29	4.23

Table 1. Impact on model performance (quality and training time) of the number of sampled Gaussians.

References

[1] Sai Bangaru, Lifan Wu, Tzu-Mao Li, Jacob Munkberg, Gilbert Bernstein, Jonathan Ragan-Kelley, Fredo Durand, Aaron Lefohn, and Yong He. Slang.d: Fast, modular and differentiable shader programming. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 42(6): 1–28, 2023. 7

[2] 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. 5

[3] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering, 2023. SIGGRAPH 2023. 8

[4] 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. 8

[5] George Kopanas and Google. slang-gaussian-rasterization: Real-time gaussian splatting rasterization using slang.d. GitHub repository, 2025. Accessed: November 20, 2025. 7

[6] Thomas Müller. tiny-cuda-nn, 2021. 7

[7] 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. 1