Stochastic Ray Tracing for the Reconstruction of 3D Gaussian Splatting

Peiyu Xu[1] Xin Sun[2] Krishna Mullia[2,3] Raymond Fei[2] Iliyan Georgiev[2] Shuang Zhao[1]
[1]University of Illinois Urbana-Champaign [2]Adobe Research [3]Canva Research

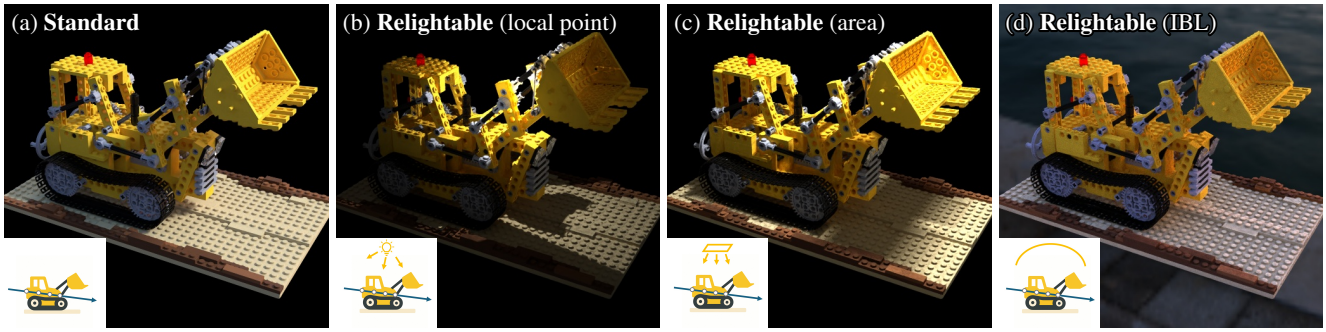

Figure 1. We introduce a differentiable stochastic formulation for ray-traced 3DGS, enabling efficient reconstruction and rendering of both *standard* and *relightable* 3DGS scenes. In this figure, we show re-renderings of our reconstructed standard 3DGS model (a) as well as relightable ones under local point light (b), area light (c), and image-based environmental illumination (d).

Abstract

Ray-tracing-based 3D Gaussian splatting (3DGS) methods overcome the limitations of rasterization—rigid pinhole camera assumptions, inaccurate shadows, and lack of native reflection or refraction—but remain slower due to the cost of sorting all intersecting Gaussians along every ray. Moreover, existing ray-tracing methods still rely on rasterization-style approximations such as shadow mapping for relightable scenes, undermining the generality that ray tracing promises.

We present a differentiable, sorting-free stochastic formulation for ray-traced 3DGS—the first framework that uses stochastic ray tracing to both reconstruct and render standard and relightable 3DGS scenes. At its core is an unbiased Monte Carlo estimator for pixel-color gradients that evaluates only a small sampled subset of Gaussians per ray, bypassing the need for sorting. For standard 3DGS, our method matches the reconstruction quality and speed of rasterization-based 3DGS while substantially outperforming sorting-based ray tracing. For relightable 3DGS, the same stochastic estimator drives per-Gaussian shading with fully ray-traced shadow rays, delivering notably higher reconstruction fidelity than prior work.

1. Introduction

3D Gaussian Splatting (3DGS) [17] represents scenes as collections of translucent 3D Gaussians, enabling real-time rendering and rapid reconstruction. Its combination of speed and expressiveness has made 3DGS a compelling choice for applications demanding both fidelity and interactivity—autonomous vehicles, immersive reality, and digital twins.

Most 3DGS methods rely on *rasterization*: individual Gaussians are projected ("splatted") onto the screen and alpha-blended to form the final image. Hardware-accelerated splatting makes this fast, but rasterization brings inherent limitations—inaccurate shadows, lacking native support for reflection or refraction, and rigid pinhole camera assumptions.

Several ray-tracing-based 3DGS methods [4, 22, 25] overcome these limitations by tracing camera rays through the scene and intersecting them with the Gaussians directly. This unlocks shadows, reflection/refraction, and nonlinear camera models (e.g., fisheye lenses) without the workarounds that rasterization demands.

However, ray-traced 3DGS comes with its own costs. Sorting all intersecting Gaussians along every camera ray is expensive, leaving these methods slower than their rasterization-based counterparts. Moreover, when handling relightable scenes, existing ray-tracing methods still fall back

on rasterization-style techniques—shadow mapping [30], deferred shading [8]—to approximate shadows and reflections, undermining the very generality that ray tracing promises.

Sun et al. [29] recently showed that stochastic ray tracing can render 3DGS scenes efficiently without sorting. Their algorithm, unfortunately, is not differentiable and therefore cannot be used for scene *reconstruction*; nor does it address *relightable* representations.

We present a **differentiable stochastic formulation** for ray-traced 3DGS—the first framework that uses stochastic ray tracing to both *reconstruct* and *render* standard and relightable 3DGS scenes. At its core is an unbiased Monte Carlo estimator for pixel-color gradients that bypasses the need for sorting Gaussians and evaluates only a small sampled subset per ray. This is particularly advantageous for relightable settings, where per-Gaussian shading (e.g., tracing shadow rays) makes exhaustive evaluation prohibitively expensive. Because our formulation is inherently ray-based, it extends naturally to shadow rays and environmental illumination, yielding physically accurate shadows and reflections without specialized rasterization passes.

Concretely, our contributions are:

1. An unbiased, sorting-free stochastic algorithm for estimating pixel-color gradients with respect to all Gaussian parameters, enabling efficient differentiable ray tracing of 3DGS scenes (Section 4.1).
2. An extension of this formulation to relightable 3DGS, where the same stochastic estimator drives shadow-ray evaluation and environment-map integration, replacing the approximate shadow-mapping pipelines used by prior work (Section 4.2).

Across standard and relightable benchmarks (Section 5), our method matches the speed of rasterization-based 3DGS while substantially outperforming sorting-based ray tracing. For relightable scenes, it delivers notably higher reconstruction fidelity, driven by per-Gaussian shading with accurate, ray-traced shadows and reflections.

2. Related Work

3DGS Foundations. 3D Gaussian Splatting (3DGS) [17] represents a scene as a set of anisotropic 3D Gaussians optimized through hardware-accelerated rasterization with differentiable compositing. This pipeline has become a standard for high-quality novel-view synthesis, and subsequent works extend it to eliminate visual artifacts [19, 27] or support non-pinhole camera models [31].

Ray Tracing for 3DGS. While rasterization is efficient, its reliance on tile-based splatting and per-pixel depth sorting limits both correctness and flexibility. 3D Gaussian Ray Tracing (3DGRT) [25] overcomes these constraints by incorporating hardware-accelerated ray tracing, enabling per-ray Gaussian sorting, arbitrary camera models, and seamless integration with mesh-based path tracing. Other works follow this direction, extending to non-Gaussian distributions [6] or proposing alternative implementations for improved efficiency or robustness [4, 22].

Efficient 3DGS Reconstruction. A large body of work seeks to accelerate 3DGS reconstruction through model compression or more effective primitive management. Compression-oriented approaches [5, 11, 14] reduce the number of Gaussians via pruning, quantization, or structure-aware simplification. SpeedySplats [14] further tightens Gaussian–tile localization to lower the per-tile primitive count. Complementary efforts improve densification and pruning heuristics for faster convergence without sacrificing quality [18, 23]. These techniques are largely orthogonal to ours: our method replaces the backward differentiation module and remains compatible with most existing acceleration strategies.

Stochastic Gaussian Splatting. A growing thread of work replaces deterministic alpha blending with stochastic transparency [9] to bypass the cost of sorting. 3DGRT introduced a biased variant that randomly retains a subset of intersected Gaussians with probability proportional to their blending weights. Sun et al. [29] developed a principled unbiased algorithm and demonstrated its effectiveness for *rendering* 3DGS assets, but did not provide a differentiable formulation suitable for *reconstruction*. StochasticSplats [19] derived a differentiable stochastic blending algorithm to address popping artifacts; however, as we show in Section 4 and the appendix, their gradient estimator suffers from high variance due to near-singular opacity terms, making it unsuitable for end-to-end 3DGS reconstruction.

Gaussian-Splatting-Based Relighting. Several works extend 3DGS to support relighting by replacing view-dependent spherical harmonics with per-Gaussian material attributes [12, 13, 21, 28], handling shadows through baked visibility or learned approximations. More recent methods [3, 10] model cast shadows via light-space splatting, achieving stronger performance on relighting benchmarks through more accurate appearance models and explicit light visibility estimation.

Our stochastic ray-tracing formulation offers a fundamentally different approach: the same per-ray visibility estimator used for primary rays extends directly to shadow rays and environmental illumination without specialized splatting passes or shadow-prediction networks, yielding physically accurate light transport within a unified framework.

3. Preliminaries

Introduced by Kerbl et al. [17], 3DGS represents a scene using a collection of 3D Gaussian primitives g_i with *mean* $\boldsymbol{\mu}_i$, *covariance* $\boldsymbol{\Sigma}_i$, *density* σ_i, and *color* c_i.

Consider a camera ray that intersects n Gaussians $g_1, g_2, \ldots, g_n$ in *arbitrary (unsorted) order*. The pixel color C is obtained by alpha blending:

$$C = \sum_{i=1}^{n} c_i \, \alpha_i \prod_{j \prec i} (1 - \alpha_j), \tag{1}$$

where $j \prec i$ denotes Gaussians g_j *in front of* (i.e., with smaller depth than) g_i, and α_i is the *opacity* of g_i:

$$\alpha_i := \sigma_i \exp\left(-(\boldsymbol{x}_i - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}_i \, (\boldsymbol{x}_i - \boldsymbol{\mu}_i)\right), \tag{2}$$

with $\boldsymbol{x}_i$ being the maximum-response point of g_i along the camera ray [25]. Directly evaluating Eq. (1)—*deterministic blending*—requires sorting all n Gaussians by depth, which remains expensive despite various acceleration efforts [5, 11, 14], especially when n is large.

Stochastic blending. Following prior work [9, 29], the pixel color C can instead be estimated stochastically. Drawing an index I with probability mass p_I and defining the random variable

$$\langle C \rangle = \frac{1}{p_I} \, c_I \, \alpha_I \prod_{j \prec I} (1 - \alpha_j), \tag{3}$$

it is straightforward to verify that $\mathbb{E}[\langle C \rangle] = C$, making $\langle C \rangle$ an *unbiased Monte Carlo estimator*[1] of C. Choosing the probability mass

$$p_I = \alpha_I \prod_{j \prec I} (1 - \alpha_j) \tag{4}$$

cancels most terms in Eq. (3), yielding the simple estimator

$$\langle C \rangle = c_I. \tag{5}$$

Sun et al. [29] realized this idea with Algorithm 1. This method examines Gaussians along the camera ray in arbitrary order, maintaining a running selection: for each Gaussian g_i, it computes the opacity α_i (Eq. (2)) and depth z_i, then replaces the current selection with g_i (i.e., $I \leftarrow i$) with probability α_i, provided g_i is closer than the current choice (i.e., $z_i < z$; see line 8). After all Gaussians have been visited, the color c of the selected Gaussian g_I is computed (line 10). This process is repeated M_f times to reduce variance.

Compared with deterministic blending, Algorithm 1 offers two key advantages: it requires *no sorting*, since Gaussians can be visited in arbitrary order; and it evaluates the color of only *one* selected Gaussian per sample—yielding significant savings when per-Gaussian shading is expensive, as we demonstrate in Section 4.

[1] Throughout this paper, we use $\langle h \rangle$ to denote a Monte Carlo estimator of h.

Algorithm 1: Monte Carlo estimate of pixel color C

1 $\langle C \rangle_\mathrm{tot} \leftarrow 0$;
2 **for** $m = 1$ *to* M_f **do**
  /* Draw index I */
3 $I \leftarrow 0$; $z \leftarrow \infty$;
4 **foreach** *Gaussian* g_i *along the camera ray* **do**
5 Compute its opacity α_i and depth z_i;
6 Draw ξ uniformly from $[0, 1)$;
7 **if** $\xi < \alpha_i$ *and* $z_i < z$ **then**
8 $I \leftarrow i$; $z \leftarrow z_i$;
  /* Obtain Gaussian color */
9 **if** $I > 0$ **then**
10 Compute the color c for the selected Gaussian g_I;
11 **else**
12 $c \leftarrow 0$;
13 $\langle C \rangle_\mathrm{tot}$ += c ; // Eq. (5)
14 **return** $\langle C \rangle_\mathrm{tot} / M_\mathrm{f}$;

4. Stochastic Reconstruction of 3DGS Scenes

Reconstructing 3DGS scenes requires differentiating pixel colors C defined in Eq. (1) with respect to the Gaussian parameters. Since the color derivatives $\partial C / \partial c_i$ and opacity derivatives $\partial C / \partial \alpha_i$ together determine all remaining parameter gradients (of $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and σ_i) via the chain rule through Eq. (2), estimating these two quantities is the core computational task.

Although stochastic blending (Algorithm 1) enables efficient *forward rendering* of 3DGS scenes, it alone is insufficient for *reconstruction*. The difficulty is that the opacity α_i governs the stochastic branching decision (line 7): naïve automatic differentiation (AD) treats this branch as fixed and therefore does not correctly differentiate through it, producing incorrect estimates of $\partial C / \partial \alpha_i$. While more sophisticated compiler techniques exist for differentiating through such discontinuities [1], their use within general-purpose GPU computation frameworks (e.g., CUDA or OptiX) remains limited in practice.

To address this challenge, we introduce a simple and efficient Monte Carlo procedure that produces *unbiased* estimates of opacity gradients (Section 4.1). We then extend this formulation with a technique for rendering and reconstructing *relightable* 3DGS scenes (Section 4.2).

4.1. Stochastic Gradient Estimation

Let $\boldsymbol{c} := (c_1, c_2, \ldots, c_n)$ and $\boldsymbol{\alpha} := (\alpha_1, \alpha_2, \ldots, \alpha_n)$ denote the colors and opacities of all n Gaussians along a camera ray. Our goal is to estimate the gradient vectors $\partial_{\boldsymbol{c}} C := (\partial C / \partial c_1, \ldots, \partial C / \partial c_n)$ and $\partial_{\boldsymbol{\alpha}} C := (\partial C / \partial \alpha_1, \ldots, \partial C / \partial \alpha_n)$.

Differentiating Eq. (1) yields their components:

$$\frac{\partial C}{\partial c_i} = \alpha_i \prod_{j \prec i} (1 - \alpha_j), \tag{6}$$

$$\frac{\partial C}{\partial \alpha_i} = \left[\prod_{j \prec i} (1 - \alpha_j) \right] \left(c_i - \sum_{k \succ i} c_k \alpha_k \prod_{i \prec t \prec k} (1 - \alpha_t) \right), \tag{7}$$

where $k \succ i$ indicates Gaussians g_k *behind* (i.e., with greater depth than) g_i; and $i \prec t \prec k$ denotes Gaussians g_t *between* g_i and g_k.

We now introduce Monte Carlo estimators $\langle \partial_{\boldsymbol{c}} C \rangle$ and $\langle \partial_{\boldsymbol{\alpha}} C \rangle$ for the gradient vectors $\partial_{\boldsymbol{c}} C$ and $\partial_{\boldsymbol{\alpha}} C$. The key idea is to reuse the same stochastic sampling from the forward pass (Algorithm 1): we draw an index $I \in \{1, 2, \ldots, n\}$ with the probability mass p_I from Eq. (4), which exactly cancels the product $\alpha_i \prod_{j \prec i} (1 - \alpha_j)$ appearing in both gradient expressions above. We then set the I-th components $\langle \partial_{\boldsymbol{c}} C \rangle_I$ and $\langle \partial_{\boldsymbol{\alpha}} C \rangle_I$ to

$$\langle \partial_{\boldsymbol{c}} C \rangle_I = 1, \tag{8}$$

$$\langle \partial_{\boldsymbol{\alpha}} C \rangle_I = \frac{1}{\alpha_I} \left(c_I - \sum_{k \succ I} c_k \alpha_k \prod_{I \prec t \prec k} (1 - \alpha_t) \right), \tag{9}$$

while leaving all the other components at zero.

Intuitively, Eq. (9) measures how much the color c_I of the selected Gaussian differs from the alpha-blended color of all Gaussians behind it—capturing the effect of "removing" g_I from the blend. However, a brute-force evaluation of this sum would require the Gaussians g_k behind the sampled g_I to be sorted. To avoid this, we apply Monte Carlo a second time, drawing an index $K \succ I$ with probability mass

$$p_{K|I} = \alpha_K \prod_{I \prec t \prec K} (1 - \alpha_t). \tag{10}$$

Note that $p_{K|I}$ has the same form as Eq. (4) but restricted to Gaussians behind g_I, so it can be sampled with the same stochastic procedure. This reduces Eq. (9) to a simple color difference:

$$\langle \partial_{\boldsymbol{\alpha}} C \rangle_I = \frac{1}{\alpha_I} (c_I - c_K). \tag{11}$$

We prove the unbiasedness of our Monte Carlo estimators in the appendix.

Monte Carlo algorithm. Algorithm 2 summarizes the complete multi-sample procedure for estimating both $\partial_{\boldsymbol{c}} C$ and $\partial_{\boldsymbol{\alpha}} C$. Each of the M_b rounds draws a Gaussian g_I (line 3–line 8) and a second Gaussian g_K behind it (line 10–line 15), as illustrated in Figure 2. The colors c^+, c^- of the two selected Gaussians are then computed (line 16–line 18), and the I-th components of the gradient estimates are updated using Eq. (8) and Eq. (11).

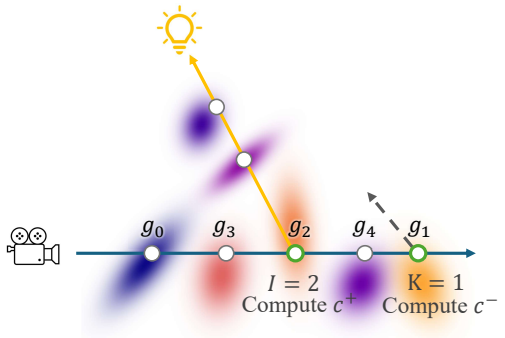

Figure 2. Our stochastic gradient estimation (Algorithm 2) works by drawing a Gaussian g_I along each camera ray (illustrated in dark blue) followed by another g_K behind it. Then, the colors c^+ and c^- of these Gaussians are computed for evaluating Eq. (11). For standard 3DGS, these colors can be obtained by evaluating the associated spherical harmonics (SH). For relightable 3DGS, on the contrary, we compute c^+ and c^- using Monte Carlo by tracing additional shadow rays (shown in yellow) toward a light source.

Algorithm 2: Our Monte Carlo estimates of pixel color gradients $\partial_{\boldsymbol{c}} C$ and $\partial_{\boldsymbol{\alpha}} C$.

1 $\langle \partial_{\boldsymbol{c}} C \rangle \leftarrow \mathbf{0}$; $\langle \partial_{\boldsymbol{\alpha}} C \rangle \leftarrow \mathbf{0}$;
2 **for** $m = 1$ *to* M_b **do**
  /* Draw index I */
3 $I \leftarrow 0$; $z \leftarrow \infty$; $\alpha \leftarrow 1$;
4 **foreach** *Gaussian* g_i *along the camera ray* **do**
5 Obtain its opacity α_i and depth z_i;
6 Draw ξ uniformly from $[0, 1)$;
7 **if** $\xi < \alpha_i$ *and* $z_i < z$ **then**
8 $I \leftarrow i$; $z \leftarrow z_i$; $\alpha \leftarrow \alpha_i$;
9 **if** $I > 0$ **then**
  /* Draw index K */
10 $K \leftarrow 0$; $z \leftarrow \infty$;
11 **foreach** *Gaussian* g_k *along the camera ray* **do**
12 Obtain its opacity α_k and depth z_k;
13 Draw ξ uniformly from $[0, 1)$;
14 **if** $z_k > z_I$ *and* $\xi < \alpha_k$ *and* $z_k < z$ **then**
15 $K \leftarrow k$; $z \leftarrow z_k$;
  /* Obtain Gaussian colors */
16 Compute the color c^+ of the Gaussian g_I;
17 **if** $K > 0$ **then**
18 Compute the color c^- of the Gaussian g_K;
19 **else**
20 $c^- \leftarrow 0$;
  /* Update the gradient estimates */
21 $\langle \partial_{\boldsymbol{c}} C \rangle_I$ += 1 ; // Eq. (8)
22 $\langle \partial_{\boldsymbol{\alpha}} C \rangle_I$ += $(c^+ - c^-)/\alpha$; // Eq. (11)
23 **return** $\langle \partial_{\boldsymbol{c}} C \rangle / M_\text{b}$, $\langle \partial_{\boldsymbol{\alpha}} C \rangle / M_\text{b}$;

Much like its forward-rendering counterpart (Algorithm 1), Algorithm 2 requires no sorting and maintains only

a minimal per-ray state—the selected indices I, K and their depths—as the ray traverses the scene in the GPU ray-tracing pipeline.

Furthermore, when we use the sorting-based forward-rendering algorithm, the sample indices I and K required for the backward pass can be collected at almost negligible cost in memory and time. In practice, we find that for simple reconstruction tasks, this approach further boosts the performance of our algorithm.

Reconstruction pipeline. Being able to stochastically compute gradients of pixel colors (Algorithm 2), we reconstruct 3DGS scenes (using multi-view input images) with a standard two-pass process as follows.

In the *forward pass*, we render images $\boldsymbol{I}$ (of a mini-batch) using the sorting-based algorithm introduced by Moenne-Loccoz et al. [25]. Meanwhile, we run our stochastic sampling algorithm (Algorithm 2) without the color computation to sample the indices I and K.

In the *backward pass*, we first compute the rendering loss $\mathcal{L}$ by comparing rendered images $\boldsymbol{I}$ against the input views, along with the pixel-wise gradient $\partial\mathcal{L}/\partial\boldsymbol{I}$. We then apply Algorithm 2 using the pre-computed indices to further backpropagate the gradients onto the color c_i and opacity α_i of each Gaussian g_i as well as its mean $\boldsymbol{\mu}_i$, covariance $\boldsymbol{\Sigma}_i$, and density σ_i via Eq. (2).

Given the resulting gradients with respect to all Gaussian parameters, we update them following the same optimization scheme as prior work [17, 25].

4.2. Handling Relightable 3DGS Scenes

The stochastic computation of pixel color gradients (Algorithm 2) can also significantly benefit the rendering and reconstruction of *relightable* 3DGS scenes. In the following, we first present a physically inspired relightable 3DGS formulation, and then discuss the rendering and reconstruction of 3D scenes under this formulation.

Relightable 3DGS formulation. In standard 3DGS, each Gaussian g_i carries a fixed color c_i—it essentially *emits* light. In a relightable formulation, c_i must instead depend on the incident illumination, since Gaussians effectively *reflect* light. Motivated by the rendering equation [16], we express the color c_i as a spherical integral over incoming directions:

$$c_i(\boldsymbol{\omega}_{\text{out}}) = \int_{\mathbb{S}^2} f_{\text{r}}(\boldsymbol{z}_i, \boldsymbol{\omega}_{\text{in}}, \boldsymbol{\omega}_{\text{out}})\, L_{\text{in}}(\boldsymbol{\omega}_{\text{in}})\, \mathrm{d}\boldsymbol{\omega}_{\text{in}}, \tag{12}$$

where $\boldsymbol{\omega}_{\text{out}}$ and $\boldsymbol{\omega}_{\text{in}}$ are the viewing and incident lighting directions, respectively. Here, f_{r} denotes the (cosine-weighted) bidirectional reflectance distribution function (BRDF), parameterized by a per-Gaussian feature $\boldsymbol{z}_i$ that can encode either physical quantities (e.g., surface albedo, roughness) or learned latent vectors for neural BRDFs.

To instantiate Eq. (12) in practice, we adopt a lightweight neural decoder Θ, shared across all Gaussians, following RNG [10]:

$$\Theta(\boldsymbol{z}_i, \boldsymbol{\omega}_{\text{in}}, \boldsymbol{\omega}_{\text{out}}, L_{\text{e}}, T) \approx f_{\text{r}}(\boldsymbol{z}_i, \boldsymbol{\omega}_{\text{in}}, \boldsymbol{\omega}_{\text{out}})\, L_{\text{in}}(\boldsymbol{\omega}_{\text{in}}), \tag{13}$$

The decoder takes as input the incident and outgoing directions $\boldsymbol{\omega}_{\text{in}}$ and $\boldsymbol{\omega}_{\text{out}}$, together with the per-Gaussian latent feature $\boldsymbol{z}_i$. To help the network learn baked-in global illumination effects such as interreflections, Θ receives two additional inputs: the direct emission L_{e} from a light source in direction $\boldsymbol{\omega}_{\text{in}}$, and the transmittance T between that light and the Gaussian g_i, defined as

$$T = 1 - \sum_{i'=1}^{n'} \alpha_{i'}^{\text{shadow}} \prod_{j' \prec i'} (1 - \alpha_{j'}^{\text{shadow}}), \tag{14}$$

where $\alpha_1^{\text{shadow}}, \ldots, \alpha_{n'}^{\text{shadow}}$ are the opacities of the unsorted Gaussians $g_1^{\text{shadow}}, \ldots, g_{n'}^{\text{shadow}}$ along the shadow ray, and $j' \prec i'$ denotes indices of Gaussians between $g_{i'}^{\text{shadow}}$ and g_i. Intuitively, the product $L_{\text{e}}\, T$ represents the direct illumination arriving at the Gaussian g_i from direction $\boldsymbol{\omega}_{\text{in}}$, attenuated by any occluding Gaussians along the way.

Rendering and training. Due to the increased integrand dimensionality, the sorting-based forward rendering algorithm becomes unaffordable for relightable Gaussians. Therefore, we switch to stochastic computation for pixel colors using Algorithm 1. In practice, for point or directional light sources, Eq. (12) reduces to a summation over directions toward individual lights (see Figure 2). For environmental illumination, on the other hand, we estimate the integral in Eq. (12) using Monte Carlo integration.

To evaluate T, we estimate the transmittance along the shadow ray using a modified version of Algorithm 1: after visiting all Gaussians $g_1^{\text{shadow}}, \ldots, g_{n'}^{\text{shadow}}$ along the shadow ray, we set the estimated transmittance to one if no Gaussian was selected (i.e., the sampled depth $z = \infty$, meaning the ray is unoccluded) and zero otherwise. A detailed description of this procedure is provided in the appendix.

Training follows the same two-pass pipeline described in Section 4.1, with one modification: the per-Gaussian color is now produced by the neural decoder Θ rather than being a stored attribute. In the backward pass, the color gradient $\langle \partial_{\boldsymbol{c}} C \rangle$ is backpropagated through Θ to update both the network weights and the per-Gaussian latent features $\boldsymbol{z}_i$, while the opacity gradient $\langle \partial_{\boldsymbol{\alpha}} C \rangle$ is backpropagated to update the shape parameters of each Gaussian as before.

Discussion. Existing relightable 3DGS methods approximate transmittance using shadow-mapping passes or costly shadow-prediction networks [3, 10]. In contrast, our stochastic ray tracing computes transmittance both efficiently and accurately, enabling faster and more faithful reconstructions—as demonstrated in Section 5. Moreover, because our formulation is inherently ray-traced, it naturally supports complex environmental illumination, whereas shadow-mapping—

Table 1. **Novel view synthesis**: Results of our approach and baselines on standard novel view synthesis benchmarks. As a **ray tracing** algorithm, our method runs at a similar speed to the rasterization-based baseline and achieves comparable quality on all benchmarks.

Method\Metric	MipNeRF360				Tanks & Temples				Deep Blending			
	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
3DGS	28.69	0.867	0.224	24m	23.14	0.853	-	14m	29.41	0.903	-	20m
3DGRT	28.40	0.862	0.233	69m	22.95	0.838	0.221	41m	29.69	0.904	0.318	54m
Ours	28.31	0.857	0.254	33m	22.57	0.830	0.221	20m	29.87	0.906	0.324	25m

Table 2. Per-iteration timing breakdown on *MipNeRF360* [2]. Our method significantly accelerates the ray tracing baseline.

Time (ms)	**Total**	**Backward**
3DGS	**31.4**	20.6
3DGRT	87.5	50.5
Ours	39.8	**17.9**

based techniques require substantial—and often imperfect—extensions to handle such scenarios.

Additionally, our rendering algorithm is agnostic to the choice of reflectance model f_{r}, and can therefore be combined with other per-Gaussian reflectance representations (e.g., GS^3 [3], Relightable-3DGS [12], GS-IR [21]).

Relation to StochasticSplats. StochasticSplats [19] also estimates gradients of the alpha-blended pixel colors C stochastically but operates within a rasterization framework rather than ray tracing. Their gradient estimator could serve as an alternative to ours; however, as we show through visual comparisons in Section 5 and a detailed analysis in the appendix, our estimator yields superior reconstruction quality.

5. Results

We evaluate our method against state-of-the-art baselines on standard benchmarks for both novel view synthesis and relightable reconstruction.

Our implementation builds on the 3DGRT codebase [25], using OptiX [26] for hardware-accelerated ray tracing. Sampling is performed in an OptiX *any-hit* program, and the forward and backward passes run inside *ray-gen* programs. Neural network evaluation and backpropagation use OptiX's Cooperative Vector interface [7]. All experiments are conducted on an Nvidia RTX 5880 Ada Generation GPU.

We set $M_{\mathrm{b}} = 8$ backward samples for all experiments. Since our stochastic algorithm tends to produce more Gaussians, we increase the densification interval to 400 in the novel view synthesis experiments to ensure a fair comparison. Multiple forward samples are drawn via independent trials within a single BVH traversal for maximum efficiency; for relighting, we use $M_{\mathrm{f}} = 15$.

By replacing sorted alpha blending with our stochastic estimator, we achieve substantially faster optimization with comparable visual quality on novel view synthesis benchmarks. Our method also naturally extends to relightable Gaussians, where accurate light transport estimation yields clear improvements over baselines.

5.1. Novel View Synthesis

Baselines. Many recent works accelerate 3DGS optimization while preserving quality. Since our method only replaces the backward differentiation module, these techniques are largely orthogonal; we therefore compare against vanilla 3DGS and 3DGRT baselines.

For fair evaluation, all reconstructed scenes are rendered with 3DGRT's deterministic alpha blending.

Datasets. We evaluate on four standard datasets: *MipNeRF-360* [2], *Tanks & Temples* [20], *Deep Blending* [15], and *NeRF Synthetic* [24].

From *MipNeRF-360*, we select four indoor scenes (*room*, *counter*, *kitchen*, *bonsai*) and three outdoor scenes (*bicycle*, *garden*, *stump*). From *Tanks & Temples* we use *train* and *truck*; from *Deep Blending*, *playroom* and *drjohnson*.

Results. Quantitative results are shown in Table 1. Our method delivers a substantial speed-up over 3DGRT while matching the speed of rasterization-based 3DGS, and achieves comparable reconstruction quality across all benchmarks. We test several forward sample counts in Table 1 and adopt 30 spp for the remaining experiments as the best speed–quality trade-off.

Figure 5 presents an equal-time visual comparison: all methods run for the same wall-clock budget. Figure 3 plots PSNR against optimization time, confirming that our method consistently converges faster than the sorted ray tracing baseline.

A per-stage timing breakdown appears in Table 2. Relative to 3DGS, our main overhead is BVH construction; relative to 3DGRT, our stochastic backward pass cuts iteration time by more than half.

StochasticSplats comparison. We re-implement StochasticSplats in our ray tracing backend, denoting this variant *SS-Tracing*. Figure 4 compares novel-view synthesis results on the same datasets; quantitative metrics are reported in the appendix.

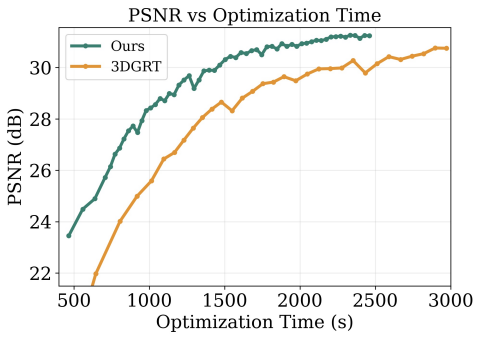

Figure 3. PSNR vs. optimization time for sorted alpha blending (3DGRT) and our method.

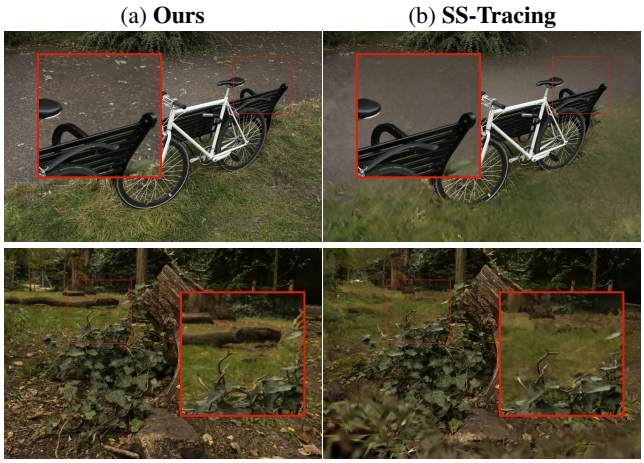

Figure 4. Reconstruction quality comparison between our method and StochasticSplats [19].

Table 3. Results of our method and baselines on the *NRHints* dataset [32].

PSNR↑ | SSIM↑	**Ours**	RNG	GS^3
Lego	**30.40**|**0.949**	26.72|0.924	26.62|0.923
Basket	**28.02**|**0.956**	19.97|0.853	23.22|0.936
Pixiu	**30.89**|0.936	30.35|**0.941**	30.38|0.937
Hotdog	**31.88**|0.955	30.38|**0.960**	25.40| 0.949
FurBall	**33.69**|**0.949**	27.82|0.926	26.36|0.931
Cat	**28.42**|0.870	28.39 |**0.888**	26.09 |0.882

5.2. Relightable Gaussian Splatting

We compare against state-of-the-art relightable baselines RNG and GS^3.

Following RNG, we adopt a two-stage pipeline. Stage 1 optimizes the Gaussians as standard 3DGS with view-dependent spherical harmonics for 15,000 iterations to initialize geometry. Stage 2 switches to the neural appearance model of Eq. (13) and fine-tunes for 85,000 iterations. The network has 4 hidden layers of dimension 64, and each Gaussian stores a 16-dimensional latent feature. Stage 1 hyperparameters match 3DGRT's novel view synthesis configuration; in Stage 2, densification, pruning, and density reset intervals are set to 3,000, 1,000, and 12,000 respectively, with all other settings unchanged.

We evaluate on the NRHints dataset [32], consistent with the baselines. Table 3 reports quantitative metrics; Figure 6 visualizes reconstruction quality.

Because our pipeline traces exact shadow rays, it produces high-quality shadows and geometry without dedicated shadow-handling modules. As shown in Figure 6, this is consistently preferable to neural approximations.

Figure 7 shows our reconstructions relit under novel environment maps. Despite training only with point-light illumination, the results are visually plausible—and our full ray-tracing pipeline incurs no additional overhead for environment lighting.

6. Discussion and Conclusion

Limitations and future work. Our stochastic formulation introduces variance into the gradient estimates, which we observed affects the behavior of the splitting and pruning heuristics used for Gaussian densification. A more detailed investigation of how the densification scheme interacts with this variance is left to future work.

Conclusion. We presented a differentiable, sorting-free stochastic ray tracing framework for 3D Gaussian Splatting—the first to use stochastic ray tracing for both reconstruction and rendering of standard and relightable scenes. By replacing exhaustive sorted evaluation with an unbiased Monte Carlo estimator that samples only a small subset of Gaussians per ray, our method matches the speed and quality of rasterization-based 3DGS while substantially outperforming sorting-based ray tracing. For relightable scenes, the same estimator enables per-Gaussian shading with fully ray-traced shadow rays, delivering notably higher reconstruction fidelity than prior approaches that rely on rasterization-style approximations.

Acknowledgments

We thank the anonymous reviewers for their feedback and suggestions. This work was partially supported by NSF grant 2553564. This work started when Peiyu Xu was an intern at Adobe Research.

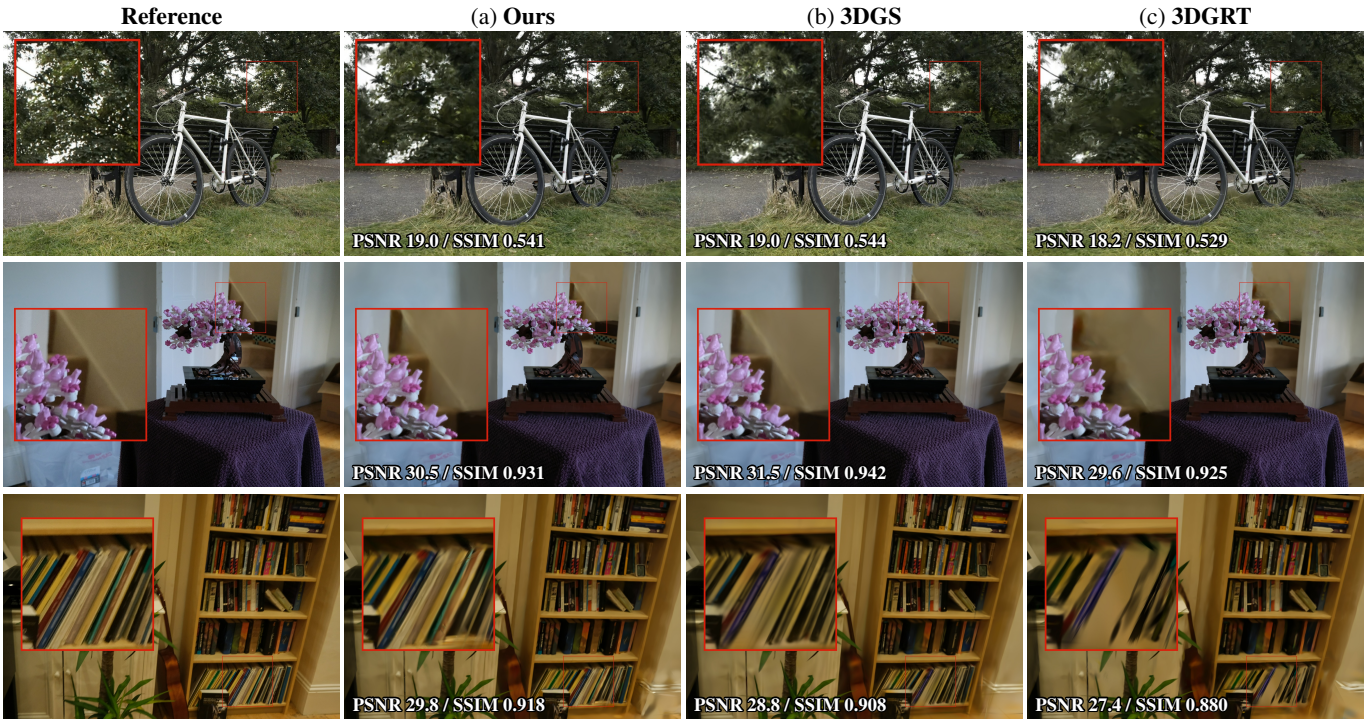

Figure 5. **Equal-time comparison** between our method and baselines. All methods run for the same wall-clock time. Our method produces comparable visual quality to 3DGS and outperforms 3DGRT.

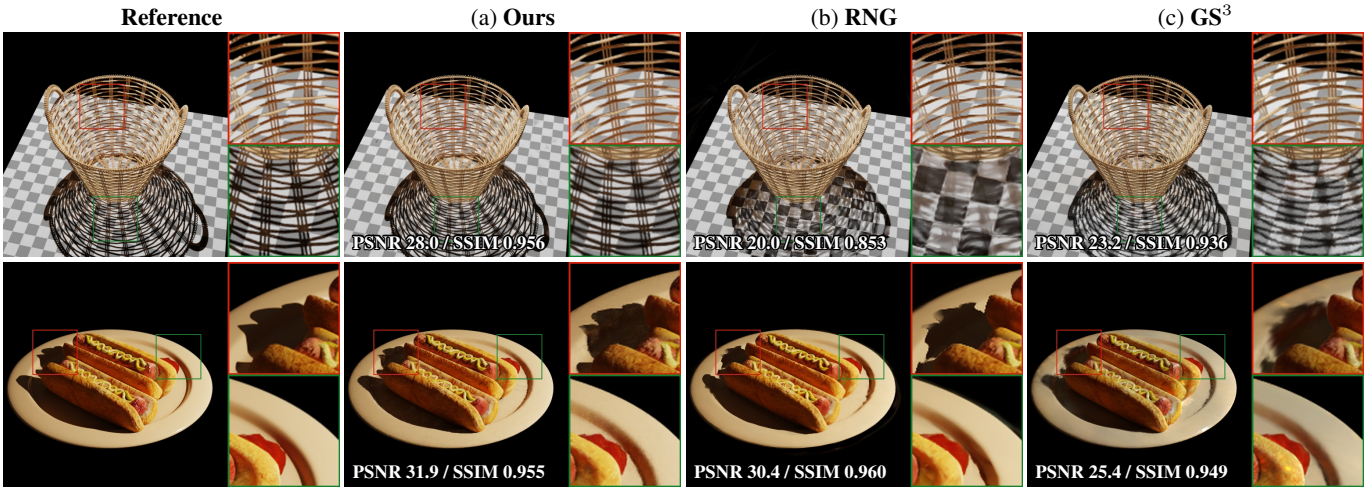

Figure 6. Reconstruction quality of our method and baselines on the relightable benchmark. Our method produces significantly better geometry, especially shadow quality.

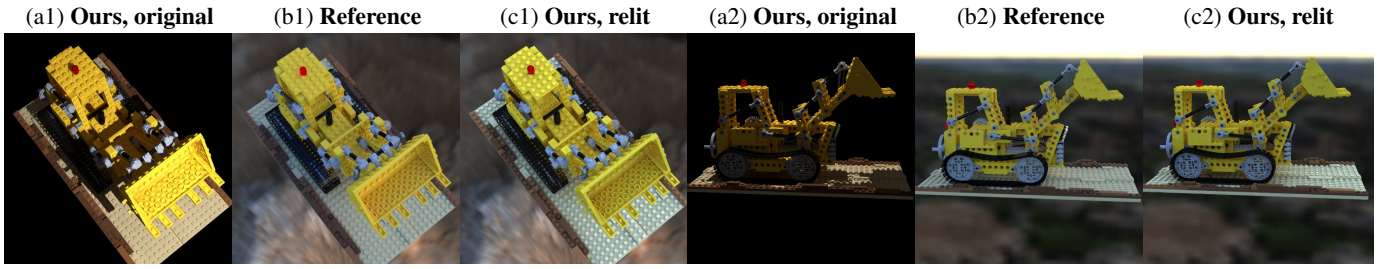

Figure 7. Re-renderings of our reconstructed 3DGS objects under novel environmental illumination.

Stochastic Ray Tracing for the Reconstruction of 3D Gaussian Splatting

Supplementary Material

A. Proof of Unbiasedness

We now provide a proof of the unbiasedness for our Monte Carlo estimators $\langle \partial_{\boldsymbol{c}} C \rangle$ and $\langle \partial_{\boldsymbol{\alpha}} C \rangle$ presented in Section 4.1 of the main paper. We recall that these estimators work by first drawing an index I and then setting the I-th components using Eq. (8) and Eq. (11) of the main paper while leaving all the other components to zero.

Then, it holds that

$$\mathbb{E}[\langle \partial_{\boldsymbol{c}} C \rangle_i] = \mathbb{P}[I = i] \cdot 1, \tag{15}$$

and

$$\begin{aligned}\mathbb{E}[\langle \partial_{\boldsymbol{\alpha}} C \rangle_i] &= \mathbb{P}[I = i] \cdot \mathbb{E}\left[\frac{c_I - c_K}{\alpha_I} \,\middle|\, I = i \right] \\ &= \mathbb{P}[I = i] \, \frac{c_i - \mathbb{E}[c_K \,|\, I = i]}{\alpha_i} \\ &= \frac{\mathbb{P}[I = i]}{\alpha_i} \left(c_i - \sum_{k \succ i} \mathbb{P}[K = k \,|\, I = i] \, c_k \right).\end{aligned} \tag{16}$$

for all $i = 1, 2, \ldots, n$.

We recall that, according to Eq. (4) and Eq. (10) of the main paper, we have

$$\mathbb{P}[I = i] = \alpha_i \prod_{j \prec i} (1 - \alpha_j), \tag{17}$$

$$\mathbb{P}[K = k \,|\, I = i] = \alpha_k \prod_{i \prec t \prec k} (1 - \alpha_t). \tag{18}$$

Substituting Eq. (17) and Eq. (18) into Eq. (15) and Eq. (16) yields

$$\mathbb{E}[\langle \partial_{\boldsymbol{c}} C \rangle_i] = \alpha_i \prod_{j \prec i} (1 - \alpha_j), \tag{19}$$

and

$$\begin{aligned}&\mathbb{E}[\langle \partial_{\boldsymbol{\alpha}} C \rangle_i] = \\ &\left(\prod_{j \prec i} (1 - \alpha_j) \right) \left(c_i - \sum_{k \succ i} c_k \alpha_k \prod_{i \prec t \prec k} (1 - \alpha_t) \right).\end{aligned} \tag{20}$$

Eq. (19) and Eq. (20) match the desired partial derivatives ${}^{\partial C}/_{\partial c_i}$ and ${}^{\partial C}/_{\partial \alpha_i}$ given by Eq. (6) and Eq. (7) of the main paper, respectively.

B. Comparison to StochasticSplats

Proposed by Kheradmand et al. [19], StochasticSplats introduces a stochastic method for estimating gradients of the alpha-blended pixel colors C with respect to the Gaussian parameters. In what follows, we present the pseudo-code for the alternative estimator in Algorithm 3, and compare the efficiency of both estimators via empirical experiments.

Algorithm 3: Monte Carlo estimator for pixel color gradient $\partial_{\boldsymbol{\alpha}} C$ introduced by StochasticSplats [19].

$\langle \partial_{\boldsymbol{\alpha}} C \rangle^{\text{ssplats}} \leftarrow \mathbf{0}$;
for $m = 1$ *to* M_b **do**
  /* Draw index I */
  $I \leftarrow 0$; $z \leftarrow \infty$; $\alpha \leftarrow 1$;
  foreach *Gaussian* g_i *along the camera ray* **do**
    Obtain its opacity α_i and depth z_i;
    Draw ξ uniformly from $[0, 1)$;
    if $\xi < \alpha_i$ *and* $z_i < z$ **then**
      $I \leftarrow i$; $z \leftarrow z_i$; $\alpha \leftarrow \alpha_i$;
  if $I > 0$ **then**
    /* Obtain Gaussian color */
    Compute the color c_I of the Gaussian g_I;
    foreach *Gaussian* g_k *along the camera ray* **do**
      Obtain its opacity α_k and depth z_k;
      if $z_k < z_I$ **then**
        /* Update the gradient estimates */
        $\langle \partial_{\boldsymbol{\alpha}} C \rangle_k^{\text{ssplats}} \mathrel{+}= {}^{-c_I}/_{(1-\alpha_k)}$; // Eq. (22)
    /* Update the gradient estimates */
    $\langle \partial_{\boldsymbol{\alpha}} C \rangle_I^{\text{ssplats}} \mathrel{+}= {}^{c_I}/_{\alpha}$; // Eq. (21)
return ${}^{\langle \partial_{\boldsymbol{\alpha}} C \rangle^{\text{ssplats}}}/_{M_\text{b}}$;

B.1. Alternative Gradient Estimator

StochasticSplats [19] uses the same procedure as our method to estimate the gradient $\partial_{\boldsymbol{c}} C$ but a different one for $\partial_{\boldsymbol{\alpha}} C$. In the following, we focus on the estimation of the latter.

As described in Algorithm 3, to estimate the gradient $\partial_{\boldsymbol{\alpha}} C := ({}^{\partial C}/_{\partial \alpha_1}, {}^{\partial C}/_{\partial \alpha_2}, \ldots, {}^{\partial C}/_{\partial \alpha_n})$ with respect to the Gaussian opacities $\alpha_1, \alpha_2, \ldots, \alpha_n$, StochasticSplats starts with drawing an index I the same way as our method, and setting

$$\langle \partial_{\boldsymbol{\alpha}} C \rangle_I^{\text{ssplats}} = \frac{c_I}{\alpha_I}. \tag{21}$$

Then, instead of randomly drawing some $K \succ I$, StochasticSplats lets

$$\langle \partial_{\boldsymbol{\alpha}} C \rangle_k^{\text{ssplats}} = \frac{-c_I}{1 - \alpha_k}, \tag{22}$$

for all Gaussians g_k located in front of the Gaussian g_I (i.e., with $k \prec I$).

Table 4. **Novel view synthesis**: Comparison between our method, our method with stochastic forward pass, and [19], in terms of both speed and quality.

Method\Metric	MipNeRF360				Tanks & Temples				Deep Blending			
	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
Ours	28.31	0.857	0.254	33m	22.57	0.830	0.221	20m	29.87	0.906	0.324	25m
Ours-FullStoch (15spp)	27.90	0.843	0.265	37m	22.37	0.821	0.233	21m	29.51	0.902	0.330	24m
Ours-Fullstoch (30spp)	28.14	0.850	0.254	52m	22.75	0.827	0.229	25m	29.75	0.904	0.327	29m
SS-Tracing	22.12	0.648	0.451	89m	14.94	0.532	0.560	54m	18.08	0.726	0.578	68m

Although this approach returns denser gradient estimates by spatting gradients to not only the I-th component but also all $k \prec I$, it suffers from two major drawbacks. First, when executed on the GPU, writing to all $k \prec I$ components in parallel requires significantly more atomic operations, which can lead to reduced computational efficiency. Second, the division by $(1 - \alpha_k)$ in Eq. (22) can produce near-infinite gradients—and therefore high variance—whenever high-opacity Gaussians make $\alpha_k \approx 1$, which occurs frequently.

The key difference between our method and [19] is the order of derivation. StochasticSplats [19] first designs a Monte Carlo estimator for alpha blending, and subsequently takes the derivative of the estimator. Our method, in contrast, first takes the derivative of alpha blending, and designs a Monte Carlo estimator for the derivative. This ordering introduces a term $1/1-\alpha_{K \prec I}$ in the density gradient, where the denominator easily evaluates to a near-0 value when an opaque Gaussian exists, causing problematic gradients.

B.2. Comparisons

To compare the efficiency of our gradient estimator (Algorithm 2 of the main paper) and the alternative one proposed by StochasticSplats (Algorithm 3), we implement the latter on our stochastic ray tracing codebase, which we refer to as *SS-Tracing*.

We give an intuition on the variance of both methods in Figure 8 by visualizing the gradient image. The images show the magnitude of screen space gradient with respect to the horizontal motion of the Gaussians, as well as a plot of the opacities of Gaussians along a ray. Our experiment shows that high opacity Gaussians frequently occur, and can easily lead to large variance for *SS-Tracing*. Empirically, it takes $16\times$ samples for SS-Tracing to reach the same level of variance as our method.

We compare the numerical metrics of our method and our re-implemented *SS-Tracing* in Table 4.

C. Additional Results

We provide additional evaluation of our method, following Section 5 of the main text.

In Table 4, we provide an ablation study of an important variant of our algorithm. We replace the sorting-based forward pass with the stochastic algorithm in Algorithm 1 of the main paper, proposed by Sun et al. [29], vary the number of samples used in the forward pass and evaluate the performance on novel view synthesis tasks. Experiment shows that our hybrid approach consistently outperforms the full stochastic variants in terms of both reconstruction quality and time.

Table 5. NeRF Synthetic Dataset

	PSNR↑	**SSIM↑**	**LPIPS↓**	**Time↓**
3DGRT	33.84	0.970	0.038	10m21s
SS-Tracing	31.67	0.958	0.050	24m31s
Ours	33.99	0.970	0.039	6m57s

In Table 5, we show the metrics of our method, 3DGRT and SS-Tracing on the NeRF-Synthetic dataset. Our performance is consistent with the results we show in the main paper.

In Figure 9, we demonstrate that our method is seamlessly compatible with distorted camera models due to the usage of ray tracing.

In Figure 10, we show additional results of the equal-time novel view synthesis benchmark. In Figure 11, we show additional results from relightable Gaussian splatting evaluations.

References

[1] Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. Systematically differentiating parametric discontinuities. *ACM Trans. Graph.*, 40(4):107:1–107:18, 2021. 3

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5470–5479, 2022. 6

[3] Zoubin Bi, Yixin Zeng, Chong Zeng, Fan Pei, Xiang Feng, Kun Zhou, and Hongzhi Wu. Gs^3: Efficient relighting with triple gaussian splatting. In *SIGGRAPH Asia 2024 Conference Papers*, 2024. 2, 5, 6

[4] Krzysztof Byrski, Marcin Mazur, Jacek Tabor, Tadeusz Dziarmaga, Marcin Kądziołka, Dawid Baran, and Przemysław Spurek. Raysplats: Ray tracing based gaussian splatting, 2025. 1, 2

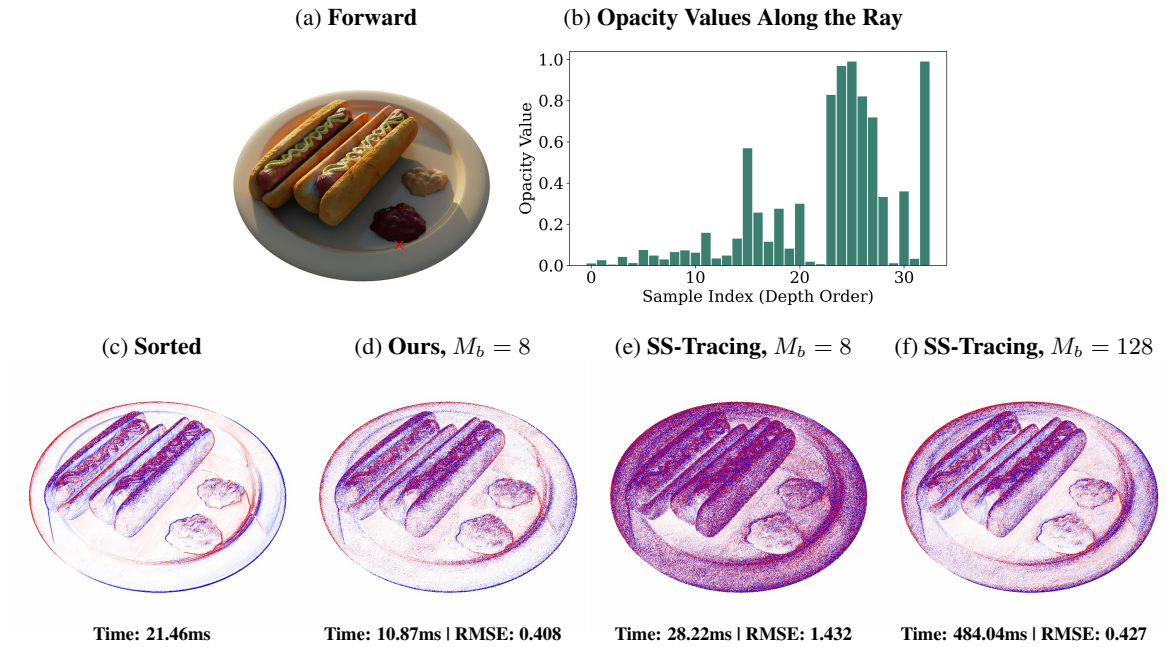

Figure 8. Visualization of the variance of the gradient obtained by our method and the baseline method. (a) shows the forward rendering result, and (b) visualizes the opacities of each Gaussian along the ray at the red cross in (a). We further visualize the gradient that the Gaussians along each pixel receive when applying: (c) Sorted alpha blending (3DGRT), (d) Ours, with $M_b = 8$, (e) Our implementation of [19], with $M_b = 8$, and (f) Our implementation of [19], with $M_b = 128$. The time for the backward pass and RMSE are provided with each image.

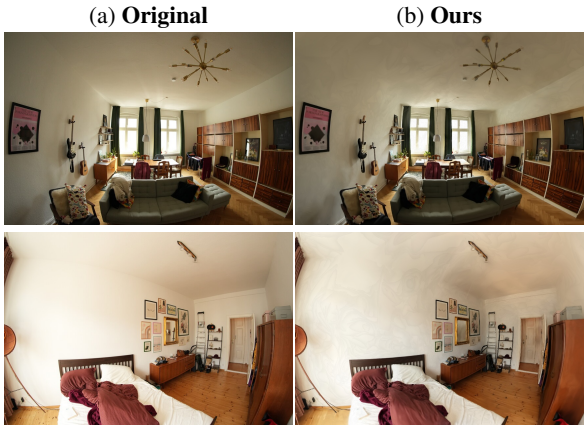

Figure 9. Similar to our ray tracing baseline, our method also seamlessly supports distorted cameras, e.g., fisheye.

[5] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac++: Towards 100x compression of 3d gaussian splatting, 2025. 2, 3

[6] Jorge Condor, Sebastien Speierer, Lukas Bode, Aljaz Bozic, Simon Green, Piotr Didyk, and Adrian Jarabo. Don't splat your gaussians: Volumetric ray-traced primitives for modeling and rendering scattering and emissive media. *ACM Trans. Graph.*, 44(1), 2025. 2

[7] NVIDIA Corporation. Optix 9.0 programming guide — cooperative vectors interface, 2025. Feature: Cooperative Vector API for ray-tracing kernels on NVIDIA RTX devices. 6

[8] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. *ACM SIGGRAPH Computer Graphics*, 22(4):21–30, 1988. 2

[9] Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. Stochastic transparency. In *I3D '10: Proceedings of the 2010 Symposium on Interactive 3D Graphics and Games*, pages 157–164, New York, NY, USA, 2010. ACM. 2, 3

[10] Jiahui Fan, Fujun Luan, Jian Yang, Milos Hasan, and Beibei Wang. Rng: Relightable neural gaussians. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2, 5

[11] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*, 2024. 2, 3

[12] Jian Gao, Chun Gu, Youtian Lin, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. Relightable 3d gaussian: Real-time point cloud relighting with brdf decomposition and ray tracing, 2023. 2, 6

[13] Chun Gu, Xiaofei Wei, Zixuan Zeng, Yuxuan Yao, and Li Zhang. Irgs: Inter-reflective gaussian splatting with 2d gaussian ray tracing, 2024. 2

[14] Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-splat: Fast 3d gaussian splatting with sparse pixels and sparse primitives. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pages 21537–21546, 2025. 2, 3

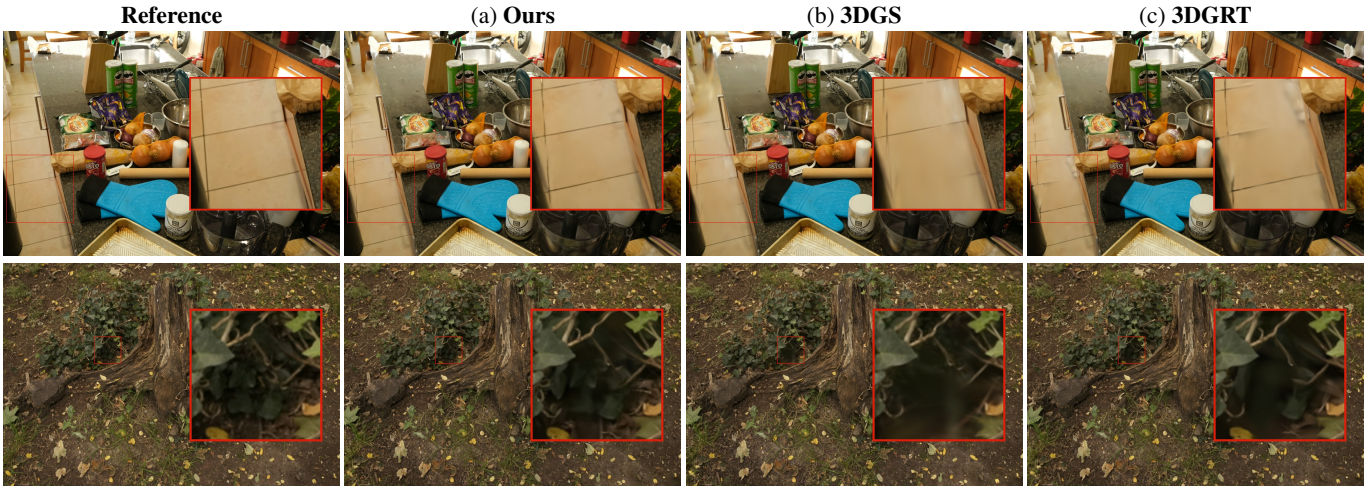

Figure 10. **Equal-time Comparison** between our method and baselines. We run the methods for the same period of wall-clock time, and compare the reconstruction quality. Our method produces comparable visual quality to 3DGS, and outperforms 3DGRT.

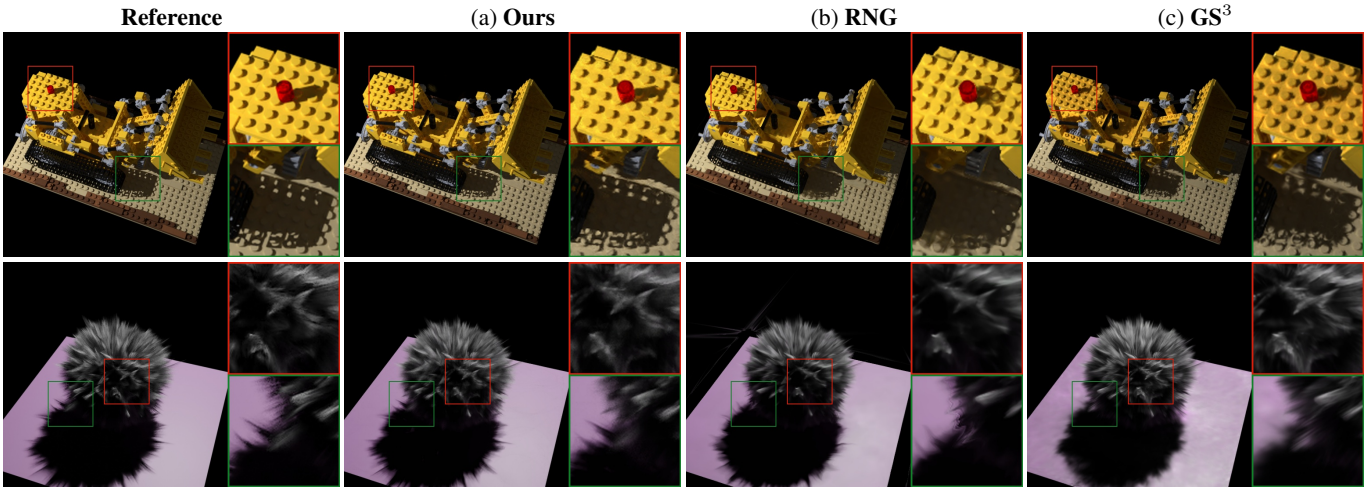

Figure 11. Visualization of reconstruction quality of our method and baselines. Our method produces significantly better geometry, and in particular, shadow quality.

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

[16] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, 1986. 5

[17] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42 (4), 2023. 1, 2, 5

[18] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo, 2025. 2

[19] Shakiba Kheradmand, Delio Vicini, George Kopanas, Dmitry Lagun, Kwang Moo Yi, Mark Matthews, and Andrea Tagliasacchi. Stochasticsplats: Stochastic rasterization for sorting-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 2, 6, 7, 1, 3

[20] 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. 6

[21] Zhihao Liang, Qi Zhang, Ying Feng, Ying Shan, and Kui Jia. Gs-ir: 3d gaussian splatting for inverse rendering. *arXiv preprint arXiv:2311.16473*, 2023. 2, 6

[22] 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 *SIGGRAPH Asia 2024 Conference Papers*, 2024. 1, 2

[23] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fer-

nando 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

[24] 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 *ECCV*, 2020. 6

[25] Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 3d gaussian ray tracing: Fast tracing of particle scenes. *ACM Transactions on Graphics*, 2024. 1, 2, 3, 5, 6

[26] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4), 2010. 6

[27] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. *ACM Transactions on Graphics*, 43(4), 2024. 2

[28] Hanxiao Sun, Yupeng Gao, Jin Xie, Jian Yang, and Beibei Wang. Svg-ir: Spatially-varying gaussian splatting for inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2

[29] Xin Sun, Iliyan Georgiev, Yun (Raymond) Fei, and Miloš Hašan. Stochastic Ray Tracing of Transparent 3D Gaussians. In *Eurographics Symposium on Rendering*. The Eurographics Association, 2025. 2, 3

[30] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274, 1978. 2

[31] Qi Wu, Janick Martinez Esturo, Ashkan Mirzaei, Nicolas Moenne-Loccoz, and Zan Gojcic. 3dgut: Enabling distorted cameras and secondary rays in gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 2

[32] Chong Zeng, Guojun Chen, Yue Dong, Pieter Peers, Hongzhi Wu, and Xin Tong. Relighting neural radiance fields with shadow and highlight hints. In *ACM SIGGRAPH 2023 Conference Proceedings*, 2023. 7