StochasticSplats: Stochastic Rasterization for Sorting-Free 3D Gaussian Splatting

Supplementary Material

1. Per-scene results

Table 3 and Table 4 show the times for different scenes and different rendering techniques.

2. Ablations

To better understand the design choices of our method, we conduct ablation studies examining the role of unbiased gradients and the capability of training from scratch.

2.1. Unbiased Gradients

In the main paper, we discussed the importance of ensuring unbiased gradients. Here, we show that without decorrelation, the gradients become biased, leading to degraded performance, particularly at lower SPP. Table 1 presents fine-tuned scenes after 1k iterations under different SPP settings. We observe that using the same samples for both the forward and backward passes causes a significant drop in quality, whereas employing two independent random samples substantially improves the results.

SPP	4	16	64	128	256
Correlated	18.04	21.20	25.66	27.25	28.09
Decorrelated	27.05	28.11	28.39	28.44	28.47

Table 1. **Unbiased gradients –** PSNR comparison between fine-tuning with correlated and decorrelated samples across different SPP settings.

2.2. Training From Scratch

While our main results are based on fine-tuning, even without fine-tuning, our method can still render 3DGS-trained scenes artifact-free using their original depth computation. Still, note that applying our improved pop-free depth computation, fine-tuning helps prevent visual artifacts that arise from mismatches between training and rendering. Regardless, for completeness, we report our performances when training from scratch – it also achieves competitive quality; see Table 2 for the Bicycle scene from MipNeRF360.

SPP	128	256	1024	AB
PSNR↑	25.11	25.21	25.37	25.62

Table 2. **Training from scratch –** results for the Bicycle scene from the MipNeRF360 dataset using our stochastic approach.

3. Tighter image-space bounding box

As our method requires rasterizing all Gaussians without an early termination transmittance threshold, any reduction in the number of Gaussians within each tile helps in our CUDA implementation. 3DGS projects a 3D Gaussian onto the image plane, resulting in a 2D Gaussian from which an axis-aligned bounding box around its center is computed in screen space. 3DGS uses a fix σ=3 to cut-off the Gaussian contribution. We follow Radl et al. [2] approach and use $t_O = \sqrt{2\log(\alpha/\epsilon_O)}$ with $\epsilon_o = 1/255$ in all our results, including 3DGS.

Given a 2D screen-space Gaussian, 3DGS computes the major and minor eigenvectors and their corresponding eigenvalues, λ_1 and λ_2. The larger eigenvalue provides a bound on the Gaussian's radius in screen space, scaled by t_O. The radius is then used to form an axis-aligned *square* bounding box around the center of the projected Gaussian.

However, for elongated Gaussians, this square approximation introduces unnecessary overhead in a tile-based implementation, increasing the number of per-tile-Gaussians. Instead, a tighter bounding box can be derived by accounting for both eigenvalues and their associated eigenvectors. Specifically, in each eigenvector direction $\mathbf{v}_i$, we set

$$\Delta_i \;=\; \sqrt{t_O\,\lambda_i}, \quad i = 1, 2, \tag{1}$$

In our OpenGL implementation, we also compute the bounding-box corners as described above, which yields a tighter fit, without requiring them to be axis-aligned. Moreover, by increasing the number of corner samples, we can further refine this bounding region with minimal additional effort.

4. Temporal anti-aliasing

At the first frame, we record each pixel's world-space position x and color. For each subsequent frame, these 3D points are projected into the new view to produce a warped version of the previous average frame's image. We then compare the warped 3D coordinates with those computed for the current frame. If they closely match, we blend both the color and the x values. Otherwise, the sample count for that pixel is reset. Let c_t denote the color image at time t, and μ_{t-1} be the accumulated warped average from previous frames. We update the color in the current frame by:

$$\bar{c}_t = \frac{N_{t-1}}{N_{t-1}+1}\mu_{t-1} + \frac{1}{N_{t-1}+1}c_{t-1}, \tag{2}$$

where N_{t-1} represents the number of accumulated samples up to frame $t-1$. The same operation applies to the 3D coordinates. We note this average position is well defined for surfaces, but poorly approximated in less confident

	ST@1	ST@2	ST@4	ST@8	ST@16	AB-GL	AB-CUDA
Room	1.91	3.14	5.70	16.24	20.49	13.73	5.92
Bonsai	1.55	2.47	4.43	12.58	15.64	10.13	4.50
Counter	2.08	3.62	6.70	19.39	24.10	12.24	5.75
Kitchen	2.83	4.51	7.96	22.33	27.28	17.86	7.41
Stump	4.12	4.42	5.09	8.98	10.72	35.80	8.90
Bicycle	5.13	5.48	6.27	11.89	14.00	64.14	12.44
Garden	5.13	5.62	6.83	13.77	16.11	73.19	12.07
Average	3.25	4.18	6.42	15.31	18.48	32.16	8.14

Table 3. Average rendering time on RTX 3090.

	ST@1	ST@2	ST@4	ST@8	ST@16	AB-GL	AB-CUDA
Room	0.99	1.40	2.33	6.98	8.35	8.87	3.76
Bonsai	0.83	1.16	1.89	5.40	6.39	6.42	2.89
Counter	1.03	1.59	2.75	8.33	9.84	8.20	3.83
Kitchen	1.42	2.05	3.41	9.60	11.20	11.36	4.89
Stump	2.15	2.29	2.64	4.26	5.09	24.15	6.32
Bicycle	2.71	2.88	3.35	5.63	6.50	41.17	9.08
Garden	2.79	3.01	3.66	6.78	7.62	44.74	8.46
Average	1.85	2.05	2.86	6.71	8.00	20.70	5.60

Table 4. Average rendering time on RTX 4090.

"volumetric" regions. We define new samples that are further from the warped mesh by more than a threshold τ to be occluded, and reset accumulation for those pixels.

5. Free-flight distance sampling

For completeness, we describe the free-flight distance sampling we use for volumetric intermixing in detail. Following [1], we define a volumetric density for a 3D Gaussian:

$$\sigma(\mathbf{x}) = \sigma_t \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right), \quad (3)$$

where σ_t scales the density of the entire Gaussian uniformly and $\mathbf{x}$ is a 3D position. The integral of this density along a ray $(\mathbf{o}, \mathbf{d})$ can be analytically computed:

$$\int_0^t \sigma(\mathbf{x}_t)\,\mathrm{d}t = \sigma_t \frac{\sqrt{\pi}}{\sqrt{2}c} \exp\left((a^2-b)/2\right) \cdot \left(\mathrm{erf}\left((a+tc)/\sqrt{2}\right) - \mathrm{erf}\left(a/\sqrt{2}\right)\right) \quad (4)$$

where t is the distance along the ray and we define:

$$a = \frac{(\mathbf{o}-\mu)^T\Sigma^{-1}\mathbf{d}}{c} \qquad b = (\mathbf{o}-\mu)^T\Sigma^{-1}(\mathbf{o}-\mu)$$
$$c = \sqrt{\mathbf{d}^T\Sigma^{-1}\mathbf{d}}.$$

In turn, this enables the analytical computation of the free-flight PDF $p(t) = \sigma(\mathbf{x}_t)\exp(-\int_0^t \sigma(\mathbf{x}_s)\,\mathrm{d}s)$ and its CDF, which is simply $1 - \exp(-\int_0^t \sigma(\mathbf{x}_s)\,\mathrm{d}s)$. This allows to derive an analytical inverse CDF sampling routine [1]:

$$d = \mathrm{erf}\left(\sqrt{2}a\right) - \sqrt{\frac{2}{\pi}}c\sigma_t^{-1}\exp\left((b-a^2)/2\right)\log(1-u)$$
$$t = \begin{cases} \frac{1}{c}\left(-a + \mathrm{erf}^{-1}(d)/d\right) & \text{if } d \in (-1,1) \\ \infty & \text{otherwise,} \end{cases}$$

where $u \sim \mathcal{U}(0,1)$. The distance t computed using this routine is distributed proportional to $p(t)$.

References

[1] 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

[2] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 43(4):1–17, 2024. 1