GRay: Ray Tracing 3D Gaussians Near the Speed of Splats — Supplemental Material

YOHAN POIRIER-GINTER, Université Laval, Canada and Inria, Université Côte d'Azur, France
JEAN-FRANÇOIS LALONDE, Université Laval, Canada
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France

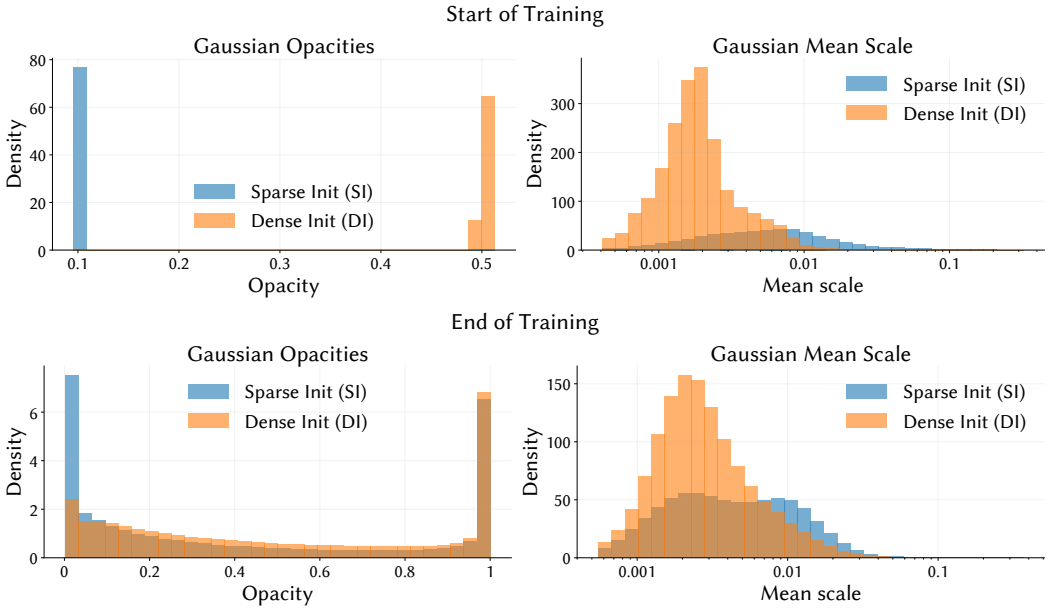

Fig. 1. Comparison of 3DGRT's Gaussian scale and opacity distributions with SI and DI, at the start and at the end of training. Values averaged over all 13 benchmarking scenes.

This supplemental material contains additional analysis, detailed ablations and additional results. Implementation details are then clarified and finally, detailed results for all individual scenes are presented.

1 Additional Analysis

This section contains additional analysis that was not presented in the main text.

1.1 Gaussian Opacity and Scale Distributions

We compared the Gaussian opacity and scale distributions between SI and DI in figure fig. 1 using 3DGRT. Interestingly, while SI initializes the Gaussian opacities low (at 0.1) and DI initializes them high (at 0.5), their distributions nearly match at the end of training. However, DI results in fewer low-opacity Gaussians. Additionally, while SI's large initial Gaussians tend to shrink a bit during training, once training ends SI still results in larger Gaussians overall.

Authors' Contact Information: Yohan Poirier-Ginter, Université Laval, Quebec, Canada and Inria, Université Côte d'Azur, Nice, France, yohan.poirier-ginter.1@ulaval.ca; Jean-François Lalonde, Université Laval, Quebec, Canada, jean-francois.lalonde@gel.ulaval.ca; George Drettakis, Inria, Université Côte d'Azur, Nice, France, george.drettakis@inria.fr.

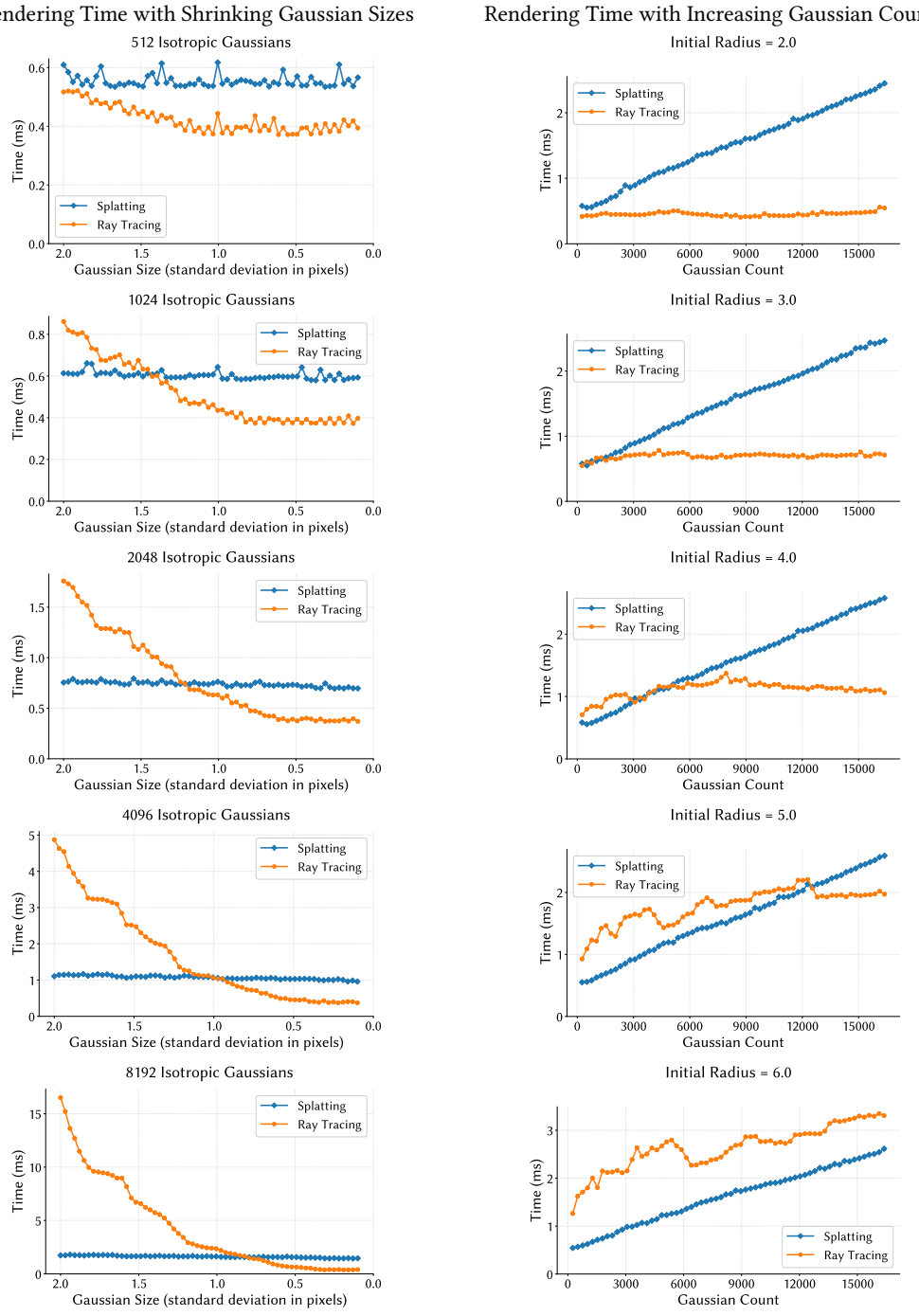

Fig. 2. Toy experiments showcasing ray tracing's ability to benefit from tiny Gaussian sizes, potentially enabling it to scale logarithmically in the Gaussian count.

1.2 Toy Experiments on Computational Complexity

To demonstrate the difference in computational complexity between splatting and ray tracing, we rendered a toy scene comprising just a few isotropic Gaussians in single 16×16 pixels tile. We used a 90 degree camera looking down on the z-plane, positioned at coordinates $(0.0, 0.0, 1.0)$, such that the camera sees a $[-1.0, 1.0] \times [-1.0, 1.0]$ square of the z-plane centered about the origin. We then placed a small number of white, zero rotation and low opacity Gaussians on this plane. We created k isotropic Gaussians per pixel that all share the same scale: as such this scale can be parameterized by a single global standard deviation value σ.

To make this experiment robust we had to control for several factors. First, early ray termination can skip many Gaussians evaluations if their opacity is high enough for the ray to go over transmittance threshold. Hence we chose a very low opacity value, 0.01, to avoid this. However, low opacities can benefit ray tracers over splatting because of adaptive clamping. To control for this we set the threshold $\alpha_{\min} = 1/255$ to match 3DGS exactly.[1] Second, ray tracing only collects Gaussians that intersect the pixel center exactly; when Gaussians are small and far away, they can occasionally fall "in between" pixels and not be rendered. To avoid this, rather than placing Gaussians randomly we fixed their positions to match the intersection of camera rays with the z plane: for each pixel, we placed k Gaussians at xy coordinates (x_i, y_j) with $x_i = 2(i + \frac{1}{2})/16 - 1$ and $y_j = 2(j + \frac{1}{2})/16 - 1$ for $i, j \in [0, 15]$. Finally, to prevent the Gaussians from overlapping exactly, we staggered their depth values slightly by setting the z-coordinate of the nth Gaussian at pixel (i, j) to $z_n = -n\epsilon$ where $\epsilon = 0.001$. We validated experimentally that each ray always intersects its own k Gaussians, at the very least.

In our first experiment, we wanted to show that splatting does not benefit from shrinking Gaussians beyond making them fit inside a single tile. As such, we set $k = 8$ to simulate a reasonable workload, yielding a total of 2048 Gaussians, and swept their standard deviation down from 2.0 all the way to 0.1. In this supplemental, we repeat the same experiment for $k \in [2, 4, 8, 16, 32]$ for reference. Results are presented in fig. 2's left column. Although which method ends is fastest depends on both the Gaussian sizes and count, these results clearly show that ray tracing is highly sensitive to Gaussian sizes while splatting is not, unless perhaps they become so large as to intersect multiple tiles. Do note however that splatting will remain faster at very low Gaussian counts, and ray tracing will require very small Gaussian sizes to surpass splatting at very high Gaussian counts.

In our second experiment, we wanted to show that ray tracing can exhibit logarithmic scaling in the number of Gaussians, provided they are small enough, while ray tracing always scales near-linearly. To this end, we selected an initial standard deviation of $\sigma = 4.0$ and progressively increased k starting from 1. When increasing k, we also decreased the standard deviation such as to approximately conserve the Gaussian's projected surface area; in other words, we used adjust values $\sigma'(k) = \sigma/\sqrt{k}$. This corresponds to halving the standard deviation each time the Gaussian count quadruples. We further validated experimentally that the effective number of ray-Gaussian stayed roughly constant as the Gaussian count increases, which is what we desired.

In this supplemental, we repeat the same experiment with the initial $\sigma \in [2.0, 3.0, 4.0, 5.0, 6.0]$ for reference. Results presented in fig. 2's right column. These results show that, while relative performance is highly sensitive to the Gaussian sizes, if Gaussian sizes are controlled for then ray tracing can exhibit log-like scaling while splatting stays linear. Bear in mind that ray racing will remain categorically slower if the initial radius is very high, and will remain categorically faster if the initial radius is very small.

[1] We did keep $n_{\text{expo}} = 2$ since we consider it core 3DGRT's proposed approach.

1.3 BVH Details and Impact on Performance

While ray tracing small primitives can potentially approach logarithmic complexity, BVH updates remain $O(n \log n)$. As such, our method's efficiency relies on BVH updates having a small constant factor. To keep the update times low, we use OptiX's support for incremental updates,[2] and only perform full rebuilds every 500 steps.

We measured the speed of BVH updates, forward pass, and backward pass, averaged across all test views and fruther averaged across all scenes.[3] Results in 3 show that incremental updates only take up $\approx 12\%$ of the the combined BVH update + forward pass + backward pass time during optimization. Full rebuilds are much slower, but very rare. We only made a minimal effort to optimize this code, and it is possible the updates could be made more efficient.

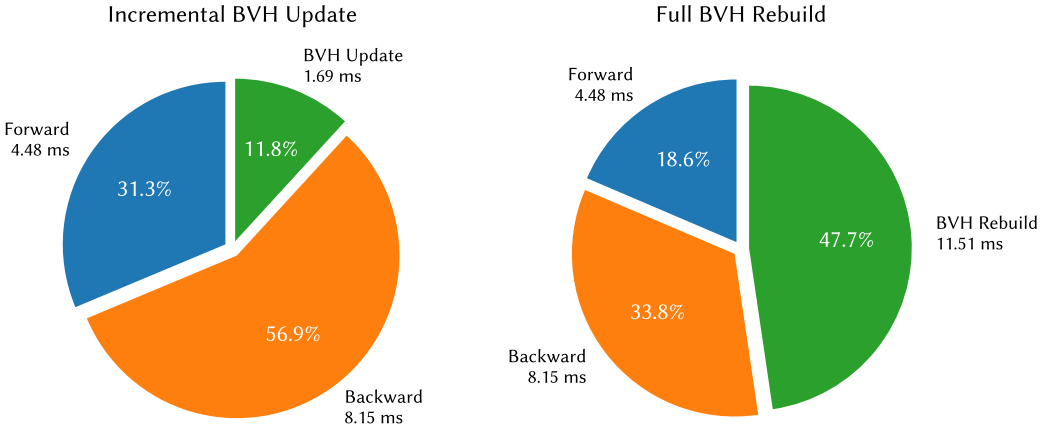

Fig. 3. Per-view training step breakdown, showing the relative importance of BVH updates compared to the forward and backward pass.

1.4 Memory Use at Higher Resolution

As discussed in the main text, working with large preallocated per-pixel linked lists is efficient, but consumes a lot of memory. Rendering very high image resolutions requires a different approach. One simple and effective strategy is tile-based rendering: the image is divided into a few large tiles, and tiles are rendered one-by-one. This keeps linked list memory use constant and remains efficient when tiles are large.

We implemented a prototype of this approach and rendered the Mip-NeRF360 Bicycle scene at increasing resolutions, from 1236×821 to 9888×6568, showing that high resolutions can be rendered with our method without running out of memory. Fig. 4 further shows that this approach remains efficient; rendering throughput actually improves at higher resolutions (ostensibly due to increased memory coherence and/or better GPU utilization).

Extending this approach to high-resolution training could be done either by training on image tiles rather than full images, or by processing the forward and backward passes tile-by-tile.

[2]Incremental updates refit the BVH internal nodes, while leaving the tree's topology unchanged.
[3]Note that these measurements were obtained by measuring the forward, backward, and update separately; the effective runtime might differ slightly.

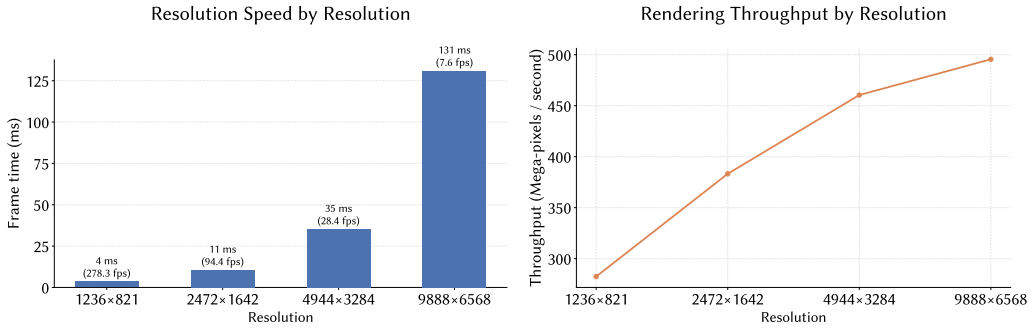

Fig. 4. Rendering speed and efficiency (throughput in megapixels per second) when rendering the Mip-NeRF360 Bicycle scene at increasing resolutions with tile-based rendering.

1.5 Explaining the Convergence Failure with DHT Disabled

When training at high transmittance thresholds without DHT, Gaussians expand pathologically, often crashing due to running out of memory; this is shown in figure 5 of the main text . DHT fixes this issue by providing a more accurate gradient for the heading Gaussians, by estimating the color of what comes behind them. Without it, these Gaussians continually expand to try cover a seemingly black background which falsely appears because of early termination. Yet expanding them further increases this problem, since bigger Gaussians intersect more pixels, while tends to make rays terminate even earlier.

2 Detailed Ablations

This section contains detailed ablations exploring the design space of ray tracing under DI, and serve to validate our design choices and final hyperparameter selection. This detailed view presents full hyperparameter sweeps and all quantitative measures for every parameter presented in the main text.

Table 1. 3DGRT BVH update times in millisecond at iteration 1 for both initialization types, comparing OBBs vs icosahedron bounds.

Method	BVH Update (ms)$_\downarrow$	#Gaussians$_\downarrow$
$3DGRT_{SI}$	**0.9**	**0.11M**
$3DGRT_{SI+Icosahedrons}$	4.5	0.11M
$3DGRT_{DI}$	**11.4**	**3.97M**
$3DGRT_{DI+Icosahedrons}$	125.7	3.97M

Table 2. Performance of 3DGRT under dense initialization with oriented bounding boxes vs. icosahedrons for bounding Gaussians.

Variant	Bound Type	Opt Time$_\downarrow$	FPS$_\uparrow$
$3DGRT_{DI}$	OBB	**00:40:05**	131
$3DGRT_{DI+Icosahedrons}$	Icosahedron	01:22:12	**145**

Restricting Gaussian Support is Faster and Worse. We first review the hyperparameters settings α_{min} and n_{expo} which 3DGRT introduced to restrict Gaussian support. Results for α_{min} are presented in tab. 3: as expected, higher values result in a marked increase in FPS counterbalanced by substantial decreased quality. Setting $\alpha_{\text{min}} = 0.001$, a very low quality, increases quality minimally over the suggested value 0.01 which we retain (PSNR 26.47 $\rightarrow$ 26.52, LPIPS 0.236 $\rightarrow$ 0.234) while also dropping FPS considerably (248 $\rightarrow$ 156); corresponding cells are highlighted in yellow.

Results for n_{expo} are shown in tab. 4. Again, selecting higher values improved speed but decreased quality noticeably, and so we retained the proposed values $n_{\text{expo}} = 2$. Matching 3DGS with $n_{\text{expo}} = 1$ clearly improves quality (PSNR 26.47 $\rightarrow$ 26.59, LPIPS 0.236 $\rightarrow$ 0.233), but also decreased FPS considerably (248 $\rightarrow$ 175); corresponding cells again highlighted in yellow.

Table 3. Quality and performance of our method under different settings of the opacity threshold α_{min}.

Variant	α_{min}	PSNR$_\uparrow$	SSIM$_\uparrow$	LPIPS$_\downarrow$	Opt. Time$_\downarrow$	FPS$_\uparrow$	Init #G$_\downarrow$	Final #G$_\downarrow$
GRay$_{\alpha_{\text{min}}=0.001}$	0.001	26.52	0.820	0.234	08:09	156	3.27M	1.70M
GRay	0.01	26.47	0.819	0.236	05:40	248	3.27M	1.52M
GRay$_{\alpha_{\text{min}}=0.02}$	0.02	26.35	0.816	0.242	04:43	313	3.27M	1.32M
GRay$_{\alpha_{\text{min}}=0.04}$	0.04	26.14	0.809	0.259	03:45	429	3.27M	1.00M
GRay$_{\alpha_{\text{min}}=0.05}$	0.05	26.03	0.805	0.270	03:27	481	3.27M	0.87M

Table 4. Quality and performance of our method under different settings of the exponent parameter n_{expo}.

Variant	n_{expo}	PSNR$_\uparrow$	SSIM$_\uparrow$	LPIPS$_\downarrow$	Opt. Time$_\downarrow$	FPS$_\uparrow$	Init #G$_\downarrow$	Final #G$_\downarrow$
GRay$_{n_{\text{expo}}=0.5}$	0.5	26.56	0.817	0.243	10:33	95	3.27M	1.50M
GRay$_{n_{\text{expo}}=1}$	1	26.59	0.821	0.233	07:05	175	3.27M	1.58M
GRay	2	26.47	0.819	0.236	05:40	247	3.27M	1.52M
GRay$_{n_{\text{expo}}=3}$	3	26.32	0.815	0.242	05:10	282	3.27M	1.44M
GRay$_{n_{\text{expo}}=4}$	4	26.28	0.812	0.247	04:55	302	3.27M	1.37M

BVH Update Times Become Critical Under DI. In Tab. 1 we compare the BVH incremental update times at initialization for both icosahedron bounds and emulated OBBs in 3DGRT. We reported averages over all test views and over all scenes, run 100 times and averaged. While update times remain negligible at <5ms with SI, they grow to over 125ms under DI becoming a substantial part of the training cost. Correspondingly, results in tab. 2 conclude that training with OBBs is two times faster than icosahedrons under DI. As such, we only considered OBBs for our ray tracer, even though frame rates with icosahedrons appear slightly higher.

Many DI Gaussians Are Useless. Tab. 5 shows the value of our binning-based merging scheme: our final setting $\psi_{\text{bin}} = 0.04$ removes over 15% of the initial Gaussians at limited cost to PSNR (26.49 $\rightarrow$ 26.47) and LPIPS (0.238 $\rightarrow$ 0.239); corresponding cells are highlighted in yellow.

Aggressive Weight-Based Pruning Maintains Good Quality. Tab. 6 compares different values of our weight-based pruning threshold γ_{prune}. The value $\gamma_{\text{prune}} = 10^{-7}$ reduces the Gaussian count to less than half (3.7M $\rightarrow$ 1.5M) while *improving* PSNR, and only worsening LPIPS by small amounts (0.234 $\rightarrow$ 0.236); corresponding cells are again highlighted in yellow. This results in substantial

Table 5. Quality and performance of our method under different settings of the bin size ψ_{bin} used for initial Gaussian binning.

Variant	ψ_{bin}	PSNR↑	SSIM↑	LPIPS↓	Opt. Time↓	FPS↑	Init #G↓	Final #G↓
$\text{GRay}_{\psi_{\text{bin}}=0.01}$	0.01	26.47	0.819	0.235	06:04	235	3.94M	1.63M
$\text{GRay}_{\psi_{\text{bin}}=0.02}$	0.02	26.48	0.819	0.235	06:01	238	3.78M	1.61M
$\text{GRay}_{\psi_{\text{bin}}=0.03}$	0.03	26.44	0.819	0.235	05:51	242	3.53M	1.58M
GRay	0.04	26.45	0.818	0.237	05:40	247	3.27M	1.52M
$\text{GRay}_{\psi_{\text{bin}}=0.05}$	0.05	26.41	0.817	0.239	05:28	255	3.03M	1.46M
$\text{GRay}_{\psi_{\text{bin}}=0.06}$	0.06	26.42	0.816	0.241	05:16	263	2.82M	1.40M
$\text{GRay}_{\psi_{\text{bin}}=0.07}$	0.07	26.35	0.815	0.245	05:04	271	2.64M	1.34M
$\text{GRay}_{\text{NoBinning}}$	–	26.49	0.819	0.234	06:23	234	3.97M	1.63M

Table 6. Quality and performance of our method under different settings of the pruning threshold γ_{prune} used for weight-based pruning.

Variant	γ_{prune}	PSNR↑	SSIM↑	LPIPS↓	Opt. Time↓	FPS↑	Init #G↓	Final #G↓
$\text{GRay}_{\gamma_{\text{prune}}=10^{-8}}$	10^{-8}	26.42	0.814	0.234	07:00	202	3.27M	2.61M
$\text{GRay}_{\gamma_{\text{prune}}=5\times10^{-8}}$	5×10^{-8}	26.47	0.817	0.234	06:09	228	3.27M	1.90M
$\text{GRay}_{\gamma_{\text{prune}}=7.5\times10^{-8}}$	7.5×10^{-8}	26.47	0.818	0.235	05:52	239	3.27M	1.69M
GRay	10^{-7}	26.47	0.819	0.236	05:40	248	3.27M	1.52M
$\text{GRay}_{\gamma_{\text{prune}}=2.5\times10^{-7}}$	2.5×10^{-7}	26.39	0.817	0.249	04:46	291	3.27M	0.97M
$\text{GRay}_{\gamma_{\text{prune}}=5\times10^{-7}}$	5×10^{-7}	26.22	0.809	0.273	04:04	340	3.27M	0.58M
$\text{GRay}_{\gamma_{\text{prune}}=10^{-6}}$	10^{-6}	25.75	0.783	0.321	03:26	414	3.27M	0.28M
$\text{GRay}_{\text{NoPruning}}$	–	26.42	0.813	0.234	07:26	197	3.27M	3.27M

improvement to FPS (157 → 248) and to optimization times (7:26 → 5:40). Our weight-based pruning significantly outperforms opacity-based pruning, which we show in sec. 3.

Table 7. Quality and performance of our method under different settings of the scale decay parameter η_{decay}.

Variant	η_{decay}	PSNR↑	SSIM↑	LPIPS↓	Opt. Time↓	FPS↑	Final #G↓	Final Avg Scale
$\text{GRay}_{\eta_{\text{decay}}=0.999800}$	0.999800	26.35	0.816	0.242	05:11	306	1.38 M	0.0148
$\text{GRay}_{\eta_{\text{decay}}=0.999825}$	0.999825	26.39	0.817	0.240	05:20	286	1.43 M	0.0153
$\text{GRay}_{\eta_{\text{decay}}=0.999850}$	0.999850	26.41	0.818	0.238	05:29	267	1.47 M	0.0159
GRay	0.999875	26.47	0.819	0.236	05:40	248	1.52 M	0.0166
$\text{GRay}_{\eta_{\text{decay}}=0.999900}$	0.999900	26.52	0.819	0.235	05:51	229	1.57 M	0.0173
$\text{GRay}_{\eta_{\text{decay}}=0.999925}$	0.999925	26.50	0.820	0.234	06:04	209	1.61 M	0.0180
$\text{GRay}_{\eta_{\text{decay}}=0.999950}$	0.999950	26.53	0.821	0.232	06:14	192	1.65 M	0.0190
$\text{GRay}_{\text{NoDecay}}$	–	26.60	0.822	0.231	06:43	157	1.73 M	0.0211

Scale Decay Efficiently Trades Quality for Speed. Results for the scale decay parameter η_{decay} are presented in tab. 7. Increasing decay improved FPS consistently and by large amounts ($\text{GRay}_{\text{NoDecay}}$@164 → GRay@259), which is explained by both reduced final Gaussian average

scales (0.0211 → 0.0166) and an increase in pruning (1.73M → 1.52M final Gaussian count); corresponding cells highlighted in yellow. However, decay also worsens quality slightly cells highlighted in blue).

Fewer Iterations can Still Converge Fully. Tab. 8 shows that it is possible to train at 15K iterations with slightly better LPIPS score as 30K iterations (0.243 → 0.236) and only a slight decrease in PSNR (0.251 → 0.247), leading to halved training times (10:24 → 05:40) almost for free; corresponding cells highlighted in yellow. We achieve this with the additional learning rate schedules described in sec. 4, which allow rapid initial growth that compensates for the reduced iteration count.

Table 8. Quality and performance of our method at reduced iteration counts, both with and without modified learning rate schedules.

Variant	Iterations	Extra LR Schedules	PSNR↑	SSIM↑	LPIPS↓	Opt. Time↓	FPS↑	Final Avg Scale ↓
GRAY$_{30K+NoSchedules}$	30000	✗	25.97	0.809	0.251	07:17	396	0.0140
GRAY$_{30K}$	30000	✓	26.51	0.817	0.243	10:24	280	0.0186
GRAY$_{15K+NoSchedules}$	15000	✗	25.71	0.806	0.251	04:06	356	0.0118
GRAY	15000	✓	26.47	0.819	0.236	05:40	248	0.0166
GRAY$_{7.5K+NoSchedules}$	7500	✗	25.21	0.799	0.261	02:12	329	0.0111
GRAY$_{7.5K}$	7500	✓	26.23	0.817	0.238	02:58	229	0.0153

DHT Stabilizes Early Ray Termination During Training. We ran our method with different levels of early ray termination with and without DHT; results are presented in tab. 9. Our method reaches slightly better LPIPS than when early termination is disabled ($\tau_{\text{min}} = 0.0$); corresponding cells are highlighted in yellow. Since these Gaussians are further skipped during the backward pass, optimization time drops significantly (6:33 → 5:40). Finally, running our method with DHT disabled shows rapid deterioration of quality metrics when truncation levels increase. For instance, GRAY$_{\text{NoDHT}+\tau_{\text{min}}=0.03}$ reaches catastrophically low PSNR (24.54) and LPIPS (0.284); corresponding cells are highlighted in blue. Certain runs at high thresholds even fail to converge due to rapid Gaussian expansions; see sec. 1.

3 Additional Results

This section contains additional results that were not presented in the main text.

3.1 Balancing Quality And Speed

Through the accumulation of techniques that increase speed with a minimal cost to quality, we were able to obtain ray-tracing optimization times comparable to 3DGS while maintaining better LPIPS. But how fast can we go, and how slow are we without these techniques? This subsection compares both extremes of the quality/speed tradeoff. Due to sometimes high memory requirements, these experiments were run at half resolution.

Low-Quality Ray Tracing. We further speed up our method by increasing the alpha threshold to $\alpha_{min} = 0.02$ and the generalized exponent to $n_{\text{expo}} = 3$, increasing the pruning threshold to $\psi_{\text{prune}} = 5\times10^{-7}$, increasing the scale decay to $\eta_{\text{decay}} = 0.9998$, and enable the SH stepping laziness at $K_{\text{lazy}} = 12$ (described in sec. 4.1). Finally, we further half training iterations to 7500. We dub this version GRay$_{\text{LQ}}$. Results in tab. 10 show that that GRay$_{\text{LQ}}$ can train in under a minute and more than double 3DGS's frame rate, at the cost of significantly reduced quality.

Table 9. Quality and performance of our method at different early ray termination transmittance thresholds τ_{min} with and without DHT. Empty rows indicate failed training runs in at least one of the scenes.

Variant	τ_{min}	DHT	PSNR$_\uparrow$	SSIM$_\uparrow$	LPIPS$_\downarrow$	Opt Time$_\downarrow$	FPS$_\uparrow$	% Skipped
GRay$_{\tau_{\text{min}}=0.001}$	0.001	✓	26.46	0.819	0.238	06:07	257	17.09
GRay$_{\tau_{\text{min}}=0.01}$	0.01	✓	26.49	0.819	0.236	05:51	255	29.38
GRay$_{\tau_{\text{min}}=0.02}$	0.02	✓	26.45	0.818	0.236	05:45	251	35.85
GRay	0.03	✓	26.47	0.819	0.236	05:40	248	40.28
GRay$_{\tau_{\text{min}}=0.04}$	0.04	✓	26.42	0.818	0.237	05:36	245	43.74
GRay$_{\tau_{\text{min}}=0.05}$	0.05	✓	26.37	0.817	0.239	05:33	243	46.50
GRay$_{\tau_{\text{min}}=0.1}$	0.1	✓	26.05	0.808	0.255	05:15	242	55.45
GRay$_{\text{NoDHT}+\tau_{\text{min}}=0.001}$	0.001	✗	26.48	0.819	0.237	06:00	269	17.08
GRay$_{\text{NoDHT}+\tau_{\text{min}}=0.01}$	0.01	✗	26.36	0.818	0.239	05:44	249	35.47
GRay$_{\text{NoDHT}+\tau_{\text{min}}=0.02}$	0.02	✗	25.60	0.809	0.257	05:38	222	47.85
GRay$_{\text{NoDHT}+\tau_{\text{min}}=0.03}$	0.03	✗	24.54	0.793	0.284	05:34	210	54.53
GRay$_{\text{NoDHT}+\tau_{\text{min}}=0.04}$	0.04	✗	-	-	-	-	-	-
GRay$_{\text{NoDHT}+\tau_{\text{min}}=0.05}$	0.05	✗	22.30	0.752	0.340	05:29	195	63.10
GRay$_{\text{NoDHT}+\tau_{\text{min}}=0.1}$	0.1	✗	-	-	-	-	-	-
GRay$_{\text{NoEarlyTerm.}}$	0.0	-	26.46	0.819	0.241	06:33	248	0.00

Table 10. Quality and performance of our ray tracer at different configuration presets (low to high quality). **Experiments run at half resolution.**

Method	Resolution	#Iterations	PSNR$_\uparrow$	SSIM$_\uparrow$	LPIPS$_\downarrow$	Init Time$_\downarrow$	Opt Time$_\downarrow$	FPS$_\uparrow$	Init #G$_\downarrow$	Final #G$_\downarrow$
GRay$_{\text{LQ}}$	1/2	7500	26.23	0.839	0.191	01:58	00:46	1051	3.27M	0.48M
GRay	1/2	15000	27.06	0.852	0.160	01:58	03:15	519	3.27M	1.47M
GRay$_{\text{HQ}}$	1/2	30000	27.41	0.847	0.154	01:58	36:11	55	3.97M	3.79M
3DGS	1/2	30000	28.02	0.875	0.166	00:00	04:22	399	0.11M	1.88M
3DGS$_{\text{DI}}$	1/2	30000	27.83	0.866	0.146	00:00	09:13	260	0.11M	3.23M

High-Quality Ray Tracing. To maximize quality, we lower the alpha threshold to $\alpha_{min} = 0.001$, the generalized exponent to $n_{\text{expo}} = 1$ (matching 3DGS), and set $\tau_{\text{min}} = 0.001$ (matching 3DGRT), disable initialization binning, and increase the pruning threshold to $\psi_{\text{prune}} = 10^{-9}$ (which nearly disables it), and finally disable scale decay entirely. We dub this version GRay$_{\text{HQ}}$. Results in tab. 10 show that that GRay$_{\text{HQ}}$ can further improve quality over GRay and approach $3DGS_{\text{DI}}$ without reaching it. In principle, it should be possible to obtain near-identical quality; we leave resolving this as future work.

3.2 Comparison to Opacity-Based Pruning

We compared our weight-based pruning to a standard opacity-based baseline, where $\gamma^{\text{prune}}_{\text{opacity}}$ denotes the opacity pruning threshold. Note that in our method higher opacity pruning thresholds are required than with 3DGS to get effective pruning, ostensibly because we do not use opacity resets. As such we ran a sweep over many different settings; results are presented in tab. 11. An LPIPS score comparable to our final solution is attained at $\gamma^{\text{prune}}_{\text{opacity}=0.075}$, but this results in significantly slower training (05:40 → 06:16) and lower final FPS (248 → 225); corresponding cells are highlighted

Table 11. Quality and performance of our method with opacity-based pruning, under different settings of the pruning threshold $\gamma^{\text{prune}}_{\text{opacity}}$ compared to our weight-based final solution.

Variant	$\gamma^{\text{prune}}_{\text{opacity}}$	PSNR$_\uparrow$	SSIM$_\uparrow$	LPIPS$_\downarrow$	Opt Time$_\downarrow$	FPS$_\uparrow$	Init #G$_\downarrow$	Final #G$_\downarrow$
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.400}}$	0.400	21.78	0.599	0.545	02:34	838	3.27M	0.01M
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.200}}$	0.200	25.65	0.782	0.329	03:07	449	3.27M	0.32M
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.150}}$	0.150	26.24	0.811	0.272	04:07	337	3.27M	0.85M
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.125}}$	0.125	26.34	0.816	0.254	04:47	291	3.27M	1.24M
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.100}}$	0.100	26.40	0.818	0.242	05:32	254	3.27M	1.69M
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.075}}$	0.075	26.45	0.818	0.235	06:16	225	3.27M	2.18M
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.050}}$	0.050	26.44	0.816	0.233	06:52	206	3.27M	2.65M
GRay$_{\gamma^{\text{prune}}_{\text{opacity}=0.025}}$	0.025	26.42	0.813	0.233	07:16	194	3.27M	3.06M
GRay	–	26.47	0.819	0.236	05:40	248	3.27M	1.52M

in yellow . Pruning more aggressively with $\gamma^{\text{prune}}_{\text{opacity}=0.075}$ results in similar training times, but a noticeable drop in quality. In short, weight-based pruning appears superior to opacity-based pruning for our method.

3.3 Variability

To verify the stability and reproducibility of our evaluation pipeline, we ran our method 3 additional times with identical hyperparameters. Results reported in tab. 12 indicate minimal variability.

Table 12. Three additional independent runs of our method.

Method	PSNR$_\uparrow$	SSIM$_\uparrow$	LPIPS$_\downarrow$	Opt Time$_\downarrow$	FPS$_\uparrow$
GRay (extra run 1)	26.45	0.818	0.237	05:40	247
GRay (extra run 2)	26.48	0.819	0.236	05:41	247
GRay (extra run 3)	26.45	0.818	0.236	05:38	247

4 Implementation Details

This section clarifies implementation and hyperparameters.

4.1 CUDA Implementation Differences

As compared to EBPRR, we do not fuse the forward and backward pass to allow for the SSIM loss and easy addition of other custom losses. We also use a CUDA-side Adam optimizer that launches a separate thread for every spherical harmonic coefficient instead of just launching one per Gaussian.

We use the fused SSIM loss proposed in Taming 3DGS [Mallick et al. 2024] as do most existing works. In experiments in sec. 3.1, we also use the lazy stepping for non-DC Spherical Harmonic (SH) bands they propose, although we leave it disabled by default; this technique consists of only taking optimization steps every K_{lazy} iterations (Taming 3DGS uses $K_{\text{lazy}} = 12$).

4.2 Learning Rate Schedules

For the Gaussian mean parameters, 3DGS employs an exponential (log-linear) learning-rate schedule of the form

$$\exp\big((1-t)\log(\lambda_{\text{init}}) + t\log(\lambda_{\text{final}})\big),$$

where t increases linearly from 0 at the start of training to 1 at iteration 30000 and where λ_{init} is the learning rate's initial value and λ_{final} its final value. We first adjusted this schedule so that $t = 1$ is reached at the final training iteration regardless of its value; for example, at iteration 15000 when training for 15000 iterations. We further introduce analogous schedules for the Gaussian scale ($0.02 \rightarrow 0.005$), rotation ($0.004 \rightarrow 0.001$), and SH DC coefficients ($0.04 \rightarrow 0.0025$); in each of these cases, the final learning rates were left unchanged, while the initial rates were increased.

4.3 Additional Hyperparameter Details

We opted for an initial opacity of 0.1, matching 3DGS rather than the configuration proposed by EDGS (0.5). We used 0.5 for other baselines under dense initialization. Our method does not prune large Gaussians based on their size, and does not feature opacity resets. We briefly experimented with adding world-space Gaussian pruning to our method, but this resulted in lower quality metrics. As such, when running dense initialization for 3DGS, we also opted to exclude this form of pruning, which also matches the configuration from 3DGRT. Note however that EDGS further excludes opacity resets, and includes additional hyperparameter changes: an opacity decay (scaling all Gaussian opacities down by 0.99 every 10 iterations) and a modified positional learning rate (clamping its value to at most the value obtained at step 8000). We did not incorporate these changes in any other method. We briefly experimented with disabling opacity resets for $3DGS_{DI}$ but this did not seem to affect quality much. The exact configuration used for all methods is clarified in tab. 13.

Table 13. Clarification of additional hyperparameter settings across all methods.

	Ours	$3DGS_{SI}$	$3DGS_{DI}$	EDGS	$3DGRT_{SI}$	$3DGRT_{DI}$
Viewport-size Gaussian pruning	✗	✓	✗	✗	✗	✗
World-size Gaussian pruning	✗	✓	✗	✗	✗	✗
Opacity resets	✗	✓	✓	✗	✓	✓
EDGS's modified LR + opacity decay	✗	✗	✗	✓	✗	✗

As a sanity check, we also ran 3DGS with dense initialization and EDGS's additional changes. Results are in tab. 14; $\cdot_{\text{MatchEDGS}}$ indicates such a run with these changes and opacity resets disabled such as to match EDGS closely. When they are included, our port of 3DGS with dense initialization ($3DGS_{DI+MatchEDGS}$) obtains quality metrics very close to EDGS.

Table 14. Reconstruction quality and performance of 3DGS with and without matching EDGS's additional modifications.

Method	$PSNR_{\uparrow}$	$SSIM_{\uparrow}$	$LPIPS_{\downarrow}$	Init $Time_{\downarrow}$	Opt $Time_{\downarrow}$	$FPS_{\uparrow}$	Init $\#G_{\downarrow}$	Final $\#G_{\downarrow}$	#Iterations
$3DGS_{SI}$	27.10	0.831	0.262	00:00	06:18	253	0.11M	2.25M	30000
$3DGS_{DI}$	26.97	0.827	0.226	01:58	10:09	241	3.97M	3.28M	30000
$3DGS_{SI+MatchEDGS}$	27.40	0.830	0.276	00:00	05:03	299	0.11M	1.41M	30000
$3DGS_{DI+MatchEDGS}$	27.45	0.849	0.211	01:58	07:47	311	3.97M	1.72M	30000
$EDGS_{RegularAdam}$	27.55	0.850	0.204	00:00	30:38	173	3.98M	2.26M	30000

Table 15. Near plane distance (z_{near}) used for each scene.

Scene	z_{near}
BICYCLE	2.00
BONSAI	3.00
COUNTER	1.00
DRJOHNSON	2.00
FLOWERS	1.00
GARDEN	2.00
KITCHEN	2.00
PLAYROOM	5.00
ROOM	1.00
STUMP	2.50
TRAIN	0.65
TREEHILL	3.25
TRUCK	1.45

4.4 Floater Masking with Near Clipping Planes

Dense initialization is floater-prone, and to alleviate this, we used near clipping planes to reduce their visual impact in rendered videos. We applied the same clipping planes to all baselines and our method. The near clipping planes z_{near} values we used are listed in tab. 15. To implement these for both ray tracers, we simply started tracing further along the initial ray, while for splatting, we culled based if the centroid was closer than z_{near} which can result in some negligible differences.

Please bear in mind that we did not use clipping planes when evaluating quality: all metrics throughout the paper were measured without them. Measuring PSNR with clipping planes offers negligible improvements (26.91 $\rightarrow$ 26.93 for TREEHILL at $z_{\text{near}} = 1.0$; 21.89 $\rightarrow$ 21.94 for GARDEN at $z_{\text{near}} = 2.0$).

5 Detailed Results

This section contains detailed results for all individual scenes for our method in tab. 16, for 3DGRT with SI in tab. 17, with DI in tab. 18, for 3DGS with SI in tab. 19, and with DI in tab. 20.

Table 16. Per-scene results of GRay.

Scene	PSNR↑	SSIM↑	LPIPS↓	Opt Time↓	FPS↑	Start #G↓	Final #G↓
Bicycle	24.38	0.730	0.233	05:38	263	3.60M	2.22M
Bonsai	31.68	0.942	0.202	08:00	126	3.93M	1.83M
Counter	28.76	0.910	0.213	08:15	108	3.22M	1.30M
Drjohnson	28.65	0.886	0.321	04:52	345	3.87M	1.33M
Flowers	20.01	0.536	0.357	05:24	247	2.96M	1.69M
Garden	26.91	0.858	0.118	05:47	241	3.21M	2.05M
Kitchen	30.59	0.926	0.141	07:29	151	2.04M	1.22M
Playroom	29.87	0.898	0.279	04:31	360	3.65M	1.24M
Room	30.26	0.921	0.246	06:17	214	4.22M	1.28M
Stump	26.35	0.771	0.234	05:12	262	2.37M	1.78M
Train	21.04	0.798	0.238	03:27	321	3.40M	0.93M
Treehill	21.89	0.619	0.333	05:00	250	2.42M	1.62M
Truck	23.77	0.850	0.157	03:46	331	3.60M	1.27M

Table 17. Per-scene results of $3DGRT_{SI}$.

Scene	PSNR↑	SSIM↑	LPIPS↓	Opt Time↓	FPS↑	Start #G↓	Final #G↓
Bicycle	24.61	0.741	0.260	56:30	63	0.05M	5.62M
Bonsai	31.62	0.940	0.239	61:02	72	0.21M	1.47M
Counter	28.74	0.907	0.244	64:43	53	0.16M	1.26M
Drjohnson	29.24	0.900	0.317	66:21	46	0.08M	2.05M
Flowers	21.50	0.613	0.340	49:36	78	0.04M	4.36M
Garden	26.68	0.849	0.140	51:33	64	0.14M	4.06M
Kitchen	30.71	0.924	0.155	72:43	47	0.24M	1.72M
Playroom	30.25	0.904	0.332	54:24	59	0.04M	1.22M
Room	30.55	0.916	0.281	60:36	63	0.11M	1.25M
Stump	26.22	0.767	0.257	53:20	65	0.03M	6.10M
Train	21.21	0.812	0.240	31:31	114	0.18M	2.34M
Treehill	22.38	0.625	0.377	58:15	56	0.05M	7.35M
Truck	24.27	0.867	0.176	34:37	98	0.14M	3.27M

Table 18. Per-scene results of $3DGRT_{DI}$.

Scene	PSNR↑	SSIM↑	LPIPS↓	Opt Time↓	FPS↑	Start #G↓	Final #G↓
Bicycle	24.72	0.762	0.208	31:57	140	4.22M	2.79M
Bonsai	31.46	0.940	0.195	56:27	77	4.50M	3.18M
Counter	28.40	0.907	0.204	59:09	68	4.49M	2.95M
Drjohnson	28.47	0.880	0.310	35:56	122	4.09M	2.68M
Flowers	20.79	0.605	0.316	31:39	171	3.68M	2.16M
Garden	26.79	0.856	0.119	40:05	110	4.02M	2.93M
Kitchen	30.37	0.925	0.139	69:22	58	4.50M	3.41M
Playroom	29.60	0.882	0.287	36:44	113	3.70M	2.82M
Room	30.22	0.917	0.240	56:27	74	4.49M	3.30M
Stump	26.35	0.766	0.236	26:35	164	2.70M	2.14M
Train	20.80	0.801	0.242	20:29	281	3.84M	1.19M
Treehill	21.79	0.611	0.332	28:39	158	3.01M	2.23M
Truck	24.28	0.866	0.141	27:31	172	4.38M	2.50M

Table 19. Per-scene results of $3DGS_{SI}$.

Scene	PSNR↑	SSIM↑	LPIPS↓	Opt Time↓	FPS↑	Start #G↓	Final #G↓
Bicycle	25.10	0.752	0.263	08:17	171	0.05M	4.00M
Bonsai	32.07	0.940	0.249	05:02	311	0.21M	1.17M
Counter	29.06	0.910	0.253	05:56	237	0.16M	1.05M
Drjohnson	29.30	0.905	0.312	06:26	202	0.08M	2.92M
Flowers	21.43	0.597	0.375	05:52	319	0.04M	2.51M
Garden	27.23	0.861	0.131	08:36	203	0.14M	3.38M
Kitchen	30.58	0.925	0.155	08:55	177	0.24M	1.72M
Playroom	29.76	0.907	0.316	04:40	309	0.04M	1.71M
Room	31.40	0.920	0.287	05:14	248	0.11M	1.16M
Stump	26.55	0.766	0.262	06:30	269	0.03M	3.96M
Train	21.96	0.815	0.241	04:58	282	0.18M	1.08M
Treehill	22.54	0.630	0.384	05:58	267	0.05M	2.67M
Truck	25.29	0.881	0.176	05:36	294	0.14M	1.98M

Table 20. Per-scene results of $3DGS_{DI}$.

Scene	PSNR$_\uparrow$	SSIM$_\uparrow$	LPIPS$_\downarrow$	Opt Time$_\downarrow$	FPS$_\uparrow$	Start #G$_\downarrow$	Final #G$_\downarrow$
Bicycle	24.91	0.760	0.203	07:53	261	4.22M	3.06M
Bonsai	32.07	0.941	0.206	11:09	195	4.50M	3.51M
Counter	28.97	0.912	0.205	15:19	140	4.49M	3.73M
Drjohnson	28.64	0.879	0.322	06:45	342	4.09M	4.07M
Flowers	20.71	0.595	0.317	08:29	276	3.68M	2.49M
Garden	27.34	0.866	0.111	11:55	195	4.02M	3.45M
Kitchen	31.33	0.929	0.137	16:44	131	4.50M	3.84M
Playroom	29.79	0.897	0.289	06:27	386	3.70M	3.67M
Room	31.29	0.927	0.240	10:57	192	4.49M	4.01M
Stump	26.39	0.761	0.241	06:13	309	2.70M	2.31M
Train	21.77	0.808	0.217	09:38	250	3.84M	2.55M
Treehill	21.87	0.598	0.324	07:29	278	3.01M	2.52M
Truck	25.46	0.879	0.125	12:58	176	4.38M	3.48M

References

Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. 2024. Taming 3DGS: High-Quality Radiance Fields with Limited Resources. In *SIGGRAPH Asia 2024 Conference Papers (SA '24)*. Association for Computing Machinery, New York, NY, USA, Article 2, 11 pages. doi:10.1145/3680528.3687694