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

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

Method	PSNR↑	SSIM↑	LPIPS↓	Init Time↓	Opt Time↓	FPS↑	Init #G↓	Final #G↓	#Iterations
3DGS	**27.10**	**0.831**	0.262	**00:00**	06:18	**253**	**0.11M**	2.25M	30000
3DGRT	26.77	0.828	0.258	**00:00**	55:01	68	**0.11M**	3.24M	30000
GRay	26.47	0.819	**0.236**	01:58	**05:40**	248	3.27M	**1.52M**	15000

Fig. 1. Reconstruction quality and performance of 3DGS [Kerbl et al. 2023] vs. 3DGRT [Moenne-Loccoz et al. 2024] vs. our method GRay averaged over the 13 standard 3DGS evaluation scenes; an example test view is also provided as reference. By leveraging dense initialization to its full extent, GRay runs nearly 4× faster and optimizes nearly 10× faster than 3DGRT while improving LPIPS.

3D Gaussian Splatting (3DGS) is a popular representation for radiance field reconstruction, distinguished by the rendering speed of its rasterization-based renderer. While 3D Gaussians can also be ray traced, this approach has so far been slower, with 3D Gaussian Ray Tracing (3DGRT) [Moenne-Loccoz et al. 2024] taking nearly one order of magnitude longer to optimize. To address this, we present GRay, a fast ray tracer for 3D Gaussians designed to close this performance gap and match 3DGS's speed. Our method leverages the algorithmic difference between both approaches: unlike rasterization, ray tracing evaluates only Gaussians that are actually intersected by a ray, leading to potentially logarithmic—rather than linear—scaling in the number of primitives. This property allows ray tracing to better exploit dense scenes composed of numerous

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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2026 Copyright held by the owner/author(s).
ACM 2577-6193/2026/5-ART14
https://doi.org/10.1145/3804496

tiny Gaussians, a configuration which has largely been overlooked. Notably, we show that dense initialization—which creates many small Gaussians—slows down rasterization, but instead *speeds up* ray tracing. Designed to leverage this effect, GRay renders nearly 4× faster and optimizes nearly 10× faster than 3DGRT while maintaining similar quality, and has competitive speed with 3DGS albeit at somewhat lower quality. Code is available at https://repo-sam.inria.fr/nerphys/gray.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Reconstruction**.

ACM Reference Format:
Yohan Poirier-Ginter, Jean-François Lalonde, and George Drettakis. 2026. GRay: Ray Tracing 3D Gaussians Near the Speed of Splats. *Proc. ACM Comput. Graph. Interact. Tech.* 9, 1, Article 14 (May 2026), 19 pages. https://doi.org/10.1145/3804496

1 Introduction

3D Gaussian Splatting (3DGS) [Kerbl et al. 2023] is a well-established representation for radiance field reconstruction and rendering, known for its speed. Its efficiency arises from its highly optimized CUDA tile-based rasterizer which accumulates and composites Gaussians in screen space: by sharing memory among neighboring pixels within a tile, it minimizes memory traffic, and by sorting with a consistent global ordering, it avoids redundant work. However, rasterization faces fundamental limitations: it is restricted to pinhole cameras and unable to compute arbitrary light paths. While extensions such as 3DGUT [Wu et al. 2025] can adapt rasterization to more general camera models, full ray tracing remains indispensable for future research, especially for inverse rendering and relighting [Gao et al. 2023; Gu et al. 2025; Xie et al. 2024; Yao et al. 2024]. Moreover, it provides a conceptually simpler formulation, free of popping artifacts and of the affine approximation. Unfortunately, ray tracing also forfeits many of the efficiencies that make splatting fast: sorting must be done per pixel, incurring more work, and shared memory can no longer be exploited, incurring higher memory traffic. Consequently, it is often regarded as inherently slower than splatting: the most popular ray tracing method, 3DGRT [Moenne-Loccoz et al. 2024], optimizes about 9× slower than 3DGS in our experiments.

In this work, we show that the speed gap between splatting and ray tracing can be closed by leveraging their difference in algorithmic scaling. At its core, ray tracing uses a BVH to select and evaluates exactly the set of Gaussians intersecting each pixel. As such, its cost is governed by the number of true intersections, which can yield logarithmic scaling in the number of primitives if they are sufficiently small. In contrast, 3DGS's rasterizer operates on tiles, evaluating *all* Gaussians overlapping a tile for every pixel within it; this results in substantial wasted computation for small Gaussians, and linear scaling regardless of their size.[1] This key algorithmic difference allows ray tracing to better exploit dense scenes composed of numerous small Gaussians; in principle, ray tracing should scale better than splatting at high Gaussian densities. This has largely been overlooked in existing works and has never been verified empirically.

First, we show that algorithmic scaling matters in practice by switching ray tracing to dense initialization (DI), which was recently proposed in EDGS [Kotovenko et al. 2025] as a superior alternative to Surface from Motion initialization with iterative densification. With DI, instead of progressively refining a sparse set of large Gaussians, matching networks are used to predict millions of small Gaussians densely covering all visible surfaces at the outset. While it was proposed to eliminate the densification stage and accelerates convergence, we show that DI further benefits ray tracing. Notably, while DI slows down rasterization, it instead *speeds up* ray tracing as predicted by complexity analysis. In fact, it reduces ray tracing workloads even at the start of optimization

[1]Prior work has identified this inefficiency and proposed partial remedies [Hanson et al. 2025a; Radl et al. 2024; Steiner et al. 2025; Wang et al. 2024], but ray tracing's pixel-wise selection provides a precise and principled solution.

with drastically inflated Gaussian counts. Hence, we argue that algorithmic scaling matters and that ray tracing is inherently well-suited to the DI regime.

We further propose a novel ray tracing method designed to leverage ray tracing's algorithmic scaling and exploit tiny Gaussians. Our method curates and extends 3DGRT [Moenne-Loccoz et al. 2024] and EBPRR [Poirier-Ginter et al. 2025] for dense initialization, and beyond the ray tracer itself, we also revisit the training regimen through improved initialization, pruning, learning rates and regularization. Our ray tracing solution, dubbed GRay, exploits DI by design.

In summary, our main contributions are:

- Showing that algorithmic complexity matters in Gaussian ray tracing, by scaling ray tracing to high counts of tiny Gaussians via dense initialization (DI). We are the first to clearly demonstrate DI's synergistic effect with ray tracing and suggest that Gaussian ray tracing could outscale splatting at very high Gaussian densities;
- A novel fast ray tracer for 3D Gaussians, GRay, with optimized hyperparameters and implementation, which nearly matches 3DGS's frame rate while maintaining the highest possible quality;
- An extensive investigation of ray tracing techniques in the context of DI, validated through comprehensive ablation studies and experimental analysis.

GRay achieves nearly 4× faster frame rates than 3DGRT and nearly 10× faster optimization with similar (or slightly lower) quality—that is, at speeds matching 3DGS with and without dense initialization. Code is available at https://repo-sam.inria.fr/nerphys/gray.

2 Related Work

In this section we first briefly describe 3DGS and provide an overview of its initialization and densification strategy, while comparing the latter to the dense initialization proposed by Kotovenko et al. [2025] (EDGS). We then survey prior work on ray tracing 3D Gaussians.

3D Gaussian Splatting. 3DGS [Kerbl et al. 2023] introduced a representation for radiance fields based on rasterized anisotropic 3D Gaussians. Given target images, 3DGS first estimates their camera poses using Structure From Motion [Schönberger and Frahm 2016; Ullman 1979] (SfM), then optimizes its Gaussians to match the targets. In 3DGS's tile-based rasterization, pixels are grouped into tiles of 16×16 pixels, for which the relevant Gaussians are depth-sorted once and then composited efficiently with shared memory. Its high quality and speed, as well as the explicit nature of its representation, have led to widespread adoption [Chen and Wang 2024; Kerbl 2025].

3DGS proposed initializing Gaussians from the sparse point cloud obtained from SfM's triangulated matches, which greatly improved on random initialization. In this approach, initial sizes are determined based on the average distance between the approximate 3 nearest neighbors. However, because this point cloud is sparse, areas where the sparse initialization is lacking must be "filled in" with additional Gaussians. As such, 3DGS employs adaptive density control throughout optimization: to increase detail, Gaussians with high error indicated by high positional gradients are split and cloned; to reduce waste, Gaussians with low opacity are pruned. While effective, this method requires a large number of steps (30000) to converge, and while it does detect many "dead" Gaussians, it does not catch them all. Hence, numerous followup works have since proposed alternative pruning criteria to be used during or after training [Fang and Wang 2024; Girish et al. 2024; Hanson et al. 2025a,b; Niemeyer et al. 2025; Papantonakis et al. 2024; Poirier-Ginter et al. 2025; Sabour et al. 2024].

Noting the inherent limitations of sparse initialization, Kotovenko et al. [2025] (EDGS) proposed dense initialization (DI) as an alternative. In their approach, dense matching networks (notably

RoMA [Edstedt et al. 2024]) are used to find pixel correspondences between views. These correspondences are then triangulated into 3D with the SfM camera poses as usual. To keep computational expenses reasonable, they first subsample the input images to a set of reference images, and for each reference image, only compute matches against a few nearest neighbors determined by comparing camera poses. EDGS's dense initialization only takes a few minutes to run.

Because DI's correspondences already cover most surfaces reasonably well, densification is no longer required: quality levels similar to sparse initialization become possible without it. Because densification is omitted and surfaces are better approximated from the start, convergence by iteration count is much faster, even though individual iterations can be costlier due to the high primitive count, especially before any pruning takes place.

3D Gaussian Raytracing. 3D Gaussian Raytracing (3DGRT) [Moenne-Loccoz et al. 2024] first studied hardware-accelerated ray tracing with OptiX [Parker et al. 2010] as an alternative for rendering 3D Gaussians. Their method bounds each Gaussian with a proxy triangle mesh and batches their evaluation, accumulating contributions in fixed-size groups of 16 Gaussians with repeated BVH traversals, as required to avoid register spilling. They also investigated different approaches for accelerating ray tracing, including techniques which traded some quality for speed; we explain the ones we retain directly in sec. 5. On our hardware, 3DGRT still optimizes about 9× slower than a recent version of 3DGS (fig. 1) despite quality reduction.

More recently, Poirier-Ginter et al. [2025] (EPBRR) proposed various improvements to 3DGRT's ray tracer, while also noting the apparent value of DI. However, these techniques were not tested against proper dense initialization and were not included in a full ray-tracing solution. They only offered limited comparisons to 3DGRT which excluded DI and trained at reduced iteration counts without Spherical Harmonics (SH), and were not validated individually. Their method used a crude custom implementation of DI which can be summarized as "binning the depth maps"—experiments in EDGS, on the other hand, argued against this approach. We detail all inclusions from EPBRR in sec. 5. Furthermore, our work thoroughly investigates the performance impact of their contributions under proper DI.

Other Gaussian Ray Tracers. Several other works have proposed ray-traced variants of 3D Gaussians, but none achieve speeds comparable to 3DGS.

RayGauss [Blanc et al. 2025a] attains very high quality at the cost of performance by treating 3D Gaussians as volumetric primitives and ray marching them in a NeRF-like manner [Mildenhall et al. 2020]. To this end, it introduces a slab-by-slab integration scheme and augments spherical harmonics with a spherical Gaussian encoding to better capture view-dependent color. While this yields higher PSNR than 3DGS, it comes with a 6× training slowdown and approximately 25× lower frame rates. RayGaussX [Blanc et al. 2025b] later extends this approach with several optimizations aimed at accelerating ray marching, including empty-space skipping, adaptive sampling, and improved ray coherence through Morton-order sorting. We briefly experiment with their Morton-order sorting but did not find it to bring a significant improvement when applied to our method. It also introduces a regularizer to limit anisotropic Gaussians as these are highly inefficient since they bound Gaussians by axis-aligned bounding boxes. These additions reduce training time to be comparable with 3DGRT, but frame rates remain approximately 2.5× slower. Most of these techniques are either orthogonal to our approach or inapplicable since their are designed for ray marching; for instance, we do not require anisotropy regularization due to our use of oriented bounding boxes (sec. 5). More crucially, our goals differ: while RayGauss and RayGaussX prioritize improving quality over 3DGS while accepting a performance loss, our goal is instead to match the speed of 3DGS while tolerating a modest reduction in quality.

RaySplats [Byrski et al. 2025a] notably accelerates ray tracing by capping the maximum number of Gaussians accumulated along each ray, making it possible to store all Gaussians in a fixed-size local memory array. This cap may trigger early termination even when the transmittance threshold has not yet been reached. To stabilize training under this approximation, RaySplats continues accumulating additional Gaussians beyond the cap and uses their contributions to correct the gradients of the front-most Gaussians. The technique we refer to as *detached hybrid transparency* follows the same underlying principle, but is based on EPBRR's implementation which excludes the fixed cap on the number of Gaussians and the secondary threshold; also note that the use of the per-pixel-linked-lists (sec. 5) relieves the need for fixed size arrays. While they report good reconstruction quality, the authors of RaySplats make no claims regarding rendering performance.

Sun et al. [2025] proposes stochastic subsampling to accelerate the ray tracing of pretrained 3D Gaussians, but doesn't offer a backward pass and only reaches frame rates about 3× below 3DGS.

Other works include custom ray tracing solutions for their specific needs: Condor et al. [2025] uses ray tracing for physically accurate volumetric integration of 3D Gaussians (potentially even including scattering), while using icosahedrons to bound Gaussians much like 3DGRT. They also proposed an alternative kernel function to increase compactness (we instead use the modified kernel proposed by 3DGRT). EVER [Mai et al. 2025] uses ray tracing to obtain the exact per-pixel sorting required for this improved blending algorithm, but uses axis-aligned bounding boxes and does not reach high performance ($4 - 20\times$ slower FPS than 3DGS). REdiSplats [Byrski et al. 2025b] ray traces flat 3D Gaussians aligned to mesh-based disks, which allows for controlling Gaussians by editing mesh vertices; however, they do not compare FPS or training times to existing approaches.

Finally, the concurrent work GRTX [Lee et al. 2026] proposed a hardware extension to accelerate 3DGRT-style repeated BVH traversals, and proposed using icosahedron meshes with instancing — akin to icosahedrons *inside* oriented bounding boxes.

Gaussian Ray Tracing's Applications. Finally, ray tracing Gaussians finds use in many downstream applications like inverse rendering and relighting [Gao et al. 2023; Gu et al. 2025; Xie et al. 2024; Yao et al. 2024], which for the moment tend to use custom-built solutions and often ray trace 2D Gaussian primitives. While ray tracing 2D Gaussians seems promising, we stick to 3D Gaussians in this work to compare more clearly against 3DGS/3DGRT. Besides, alternative approaches for ray tracing radiance fields have also been proposed [Chen et al. 2025; Govindarajan et al. 2025], but these are out of scope for this paper, since they introduce different representations.

3 Can Ray Tracing Surpass Splatting?

The current literature on Gaussian rendering portrays ray tracing as inherently slower than splatting, a price that must be paid for its numerous advantages. However, the two approaches are algorithmically distinct and exhibit fundamentally different scaling behavior. At its core, 3DGS's tile-based rasterizer scales linearly with the number of Gaussian primitives in a tile, since sorting involves all primitives and since each primitive is evaluated for every pixel in that tile. In contrast, ray tracing selects only the Gaussians relevant to each pixel through BVH traversal, and therefore scales with the number of true ray-Gaussian intersections rather than the number of primitives. This distinction has two important consequences. First, shrinking Gaussians consistently accelerates ray tracing by reducing intersection counts, whereas rasterization only benefits when Gaussians are large enough to cover multiple tiles. Second, when the Gaussian count grows without increasing the likelihood of intersection—namely when surfaces are represented with more numerous smaller Gaussians—ray tracing can approach logarithmic scaling in the number of primitives. As such, in the limit of large counts of tiny Gaussians, ray tracing can potentially outperform rasterization.

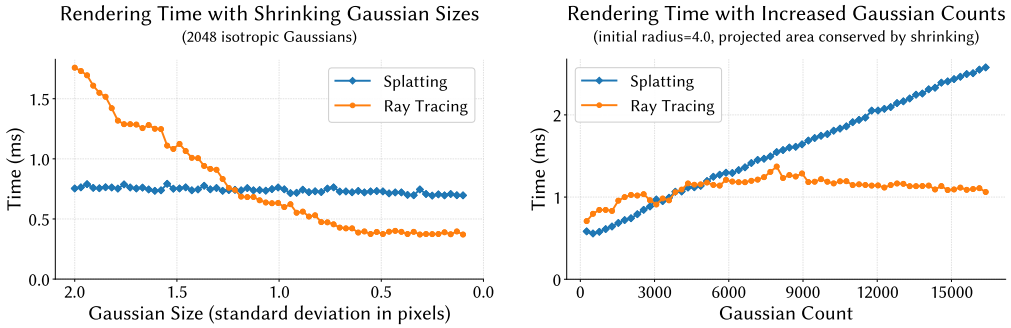

Fig. 2. Ray tracing can scale better than splatting when the rendered primitives are small. This test renders a single 16×16 pixel tile at different Gaussian counts and scales. Left: as Gaussian shrink, ray tracing speeds up to eventually overtake splatting which stays nearly constant. Right: when more Gaussians are added without increasing their total projected area, ray tracing exhibits logarithmic scaling.

Ray Tracing Speeds Up When Gaussians Shrink. To show this difference in computational complexity clearly, we rendered a single 16×16 pixel tile with both 3DGS and our ray tracer GRay, which we introduce later in this paper, and progressively reduced Gaussian sizes. The toy scene consists of a grid of 2048 isotropic Gaussians, with a 90° field of view camera looking down on the grid. We rendered a first image with Gaussians at a standard deviation of 2.0 pixels, and then progressively reduced their standard deviation down to 0.1 pixels. Results are shown in the left of fig. 2. While ray tracing initially renders much slower, as the Gaussians shrink, it can eventually overtake splatting since it benefits from smaller Gaussian sizes. Which method is fastest depends on the number of Gaussians, but the such trends are consistent across counts; see the supplemental for details.

Ray Tracing Can Scale Logarithmically. Next, we tested scaling in the number of primitives. We rendered the same toy scene at one Gaussian per pixel with standard deviations of 4.0 pixels and progressively increased Gaussian counts up to 64 per pixel. While adding Gaussians, we progressively shrunk their size such as to conserve the same projected surface area; for instance, quadrupling the Gaussian count involved halving their standard deviations. Results are shown in the right of fig. 2: when Gaussian size is controlled for, ray tracing exhibits logarithmic scaling under increased primitive counts. Hence, depending on the size of its primitives, ray tracing can eventually surpass splatting as Gaussian counts increase. Which method is fastest in practice is sensitive to the initial standard deviation, but the scaling behavior is unchanged by this initial size; see the supplemental for details. Also note that although BVH construction remains $O(n \log n)$, update cost is small in practice; this is also explained in the supplemental.

4 Algorithmic Complexity Matters in Practice

To test whether the difference in algorithmic complexity exposed in sec. 3 is relevant in practice, we employ the well-known ray tracer 3DGRT [Moenne-Loccoz et al. 2024] and switch its configuration to use dense initialization (DI) [Kotovenko et al. 2025], a realistic scenario where Gaussians are both smaller and more numerous.

DI Slows Down Splatting but Speeds Up Ray Tracing. We ran both 3DGS and 3DGRT with and without DI for just 500 steps (stopping before any densification or pruning takes place so Gaussian counts stay constant). This lets us accurately measure the performance impact of switching to DI and

Fig. 3. Different Gaussian initializations visualized with 3DGRT. From left to right: sparse (SfM) initialization, sparse initialization with Gaussian scales matching 3DGS exactly (note how primitives are very large, which is unfavorable for ray-tracing), and dense initialization.

compare both rendering methods at equal Gaussian counts. Tab. 1 gives the results, where $\cdot_{SI}$ denotes a method with sparse (SfM) initialization while $\cdot_{DI}$ denotes a method with dense initialization. 3DGRT$_{SI}^{*}$ further denotes 3DGRT$_{SI}$ with initialization adjusted to match 3DGS$_{SI}$ exactly[2]; these different initialization types are displayed in fig. 3. As expected, switching to DI slows down 3DGS's optimization by over 3× while halving its frame rate, which is explained by the Gaussian count increase from around 100K to nearly 4M. Yet strikingly, with the same Gaussian counts, dense initialization *speeds up* ray tracing instead of slowing it down, both in terms of optimization time and FPS.

Table 1. Speed of the first 500 iterations for 3DGS and 3DGRT for both initialization types, averaged over all scenes. Speedups are measured within each family using 3DGS$_{SI}$ and 3DGRT$_{SI}$ as baselines. FPS is measured at iteration 500.

Method	#Gaussians$_{\downarrow}$	Opt Time$_{\downarrow}$	FPS$_{\uparrow}$
3DGS$_{SI}$	**0.11M**	**00:16**	**660**
3DGS$_{DI}$	3.97M	00:50 (0.32×)	282 (0.43×)
3DGRT$_{SI}$	**0.11M**	00:31	96
3DGRT$_{SI}^{*}$	**0.11M**	00:30	98
3DGRT$_{DI}$	3.97M	**00:28** (1.11×)	**140** (1.46×)

DI Reduces the Ray Tracing Workload Despite Increased Gaussian Counts. We further analyzed the ray tracing workload induced by both initialization types in more detail. Our experiments show that switching to DI *reduces* the ray tracing workload—even at the start of training, when the Gaussian counts are >35× larger. Fig. 4 compares the number of hit Gaussians per ray across training alongside the required number of BVH traversals[3]: DI reduces the number of hit Gaussians by around about 3.5× (209.2 → 60.2 per pixel) and the number of traversals by around 2.75× (4.01 → 1.45 per pixel) on average across iterations. We conjecture that smaller Gaussians model surfaces more precisely, with less "fuzz" and without layering numerous mid-to-large Gaussians within object volumes. In any case, DI appears unequivocally more efficient than SI when working with ray tracing.

[2]While 3DGS uses the average distance to the approximate 3 nearest neighboring points to scale the initial Gaussians, 3DGRT instead uses the distance to the nearest camera.

[3]A hit is counted for each ray intersection with a Gaussian bounding box; hits and traversals are counted for the forward pass only, and so the effective counts are doubled when accounting for the backward pass.

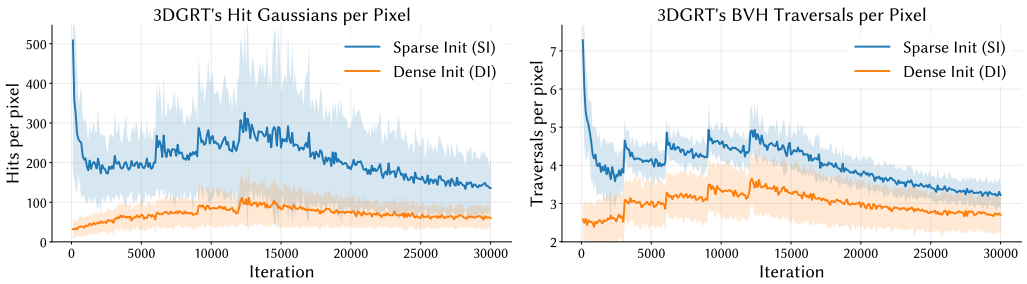

Fig. 4. Per-pixel ray–Gaussian intersection statistics over training with sparse (SI) and dense (DI) initialization. Mean hits per pixel (left) and mean BVH traversals per pixel (right), averaged across all scenes, with shaded ±1 standard deviation bands.

5 GRay: A Fast Ray-Tracing Method for 3D Gaussians

Based on these insights, we develop a novel method designed to exploit ray tracing's algorithmic scaling. We build our method on 3DGRT [Moenne-Loccoz et al. 2024], EPBRR [Poirier-Ginter et al. 2025] and EDGS [Kotovenko et al. 2025] by curating their techniques and modifying the training regimen to better exploit large counts of smaller Gaussians. We validated our design choices through detailed ablations and analysis presented in the supplemental; highlights from these experiments are also presented here directly. All metrics report averages over all thirteen 3DGS standard benchmarking scenes.[4] Code for our method will be open sourced.

Dense Initialization (DI). We use the DI method proposed by EDGS directly and with their recommended parameters: 25,000 matches per reference frame, a total of 180 reference frames, and 3 neighbors per reference. We briefly explored alternatives but did not find important differences in quality; notably, adding more Gaussians did not help. Likewise, we initialize Gaussian scales isotropically to 0.0005 times the distance to the reference camera which produced the Gaussian.

Restricting Gaussian Kernel Support. 3DGRT proposed reducing the Gaussians' footprints with two techniques. First, by replacing the Gaussian kernel with a generalized version like:

$$\exp\Big[-\frac{1}{2n^{\text{expo}}}\big((x-\mu)^T\Sigma^{-1}(x-\mu)\big)^{n_{\text{expo}}}\Big],$$

where increasing the exponent n_{expo} makes Gaussians "bulkier". Second, with adaptive clamping, i.e., by defining a Gaussian's support in terms of its alpha value after opacity multiplication, such that all alpha values falling below a threshold $\alpha_{\min}$ are zeroed. Both of these techniques decrease superfluous ray-Gaussian intersections. We employ these two techniques with their suggested hyperparameters ($n_{\text{expo}} = 2$ and $\alpha_{min} = 0.01$) since we aim to stay near 3DGRT's quality. However, 3DGRT uses $\frac{4.5}{3\cdot 2n^{\text{expo}}}$ instead of $\frac{1}{2n^{\text{expo}}}$; opting for the latter makes initialization less sensitive to the setting of n^{expo} and results in smaller initial Gaussians in practice.

Interestingly, while dynamic clamping leads to visible artifacts for large Gaussians, they can be less apparent when working with DI since most Gaussians stay small.

Oriented Bounding Boxes (OBBs). We found OBBs essential when working with DI since the Gaussian counts start much higher, making BVH update times more important. 3DGRT initially proposed using triangle icosahedrons to bound Gaussians, but they have slow BVH updates. This

[4]All experiments were performed on a single NVIDIA GeForce RTX 4090.

becomes problematic when dealing with millions of tiny Gaussians, since updating them at each training iteration requires transforming 12 vertices for each icosahedron. EPBRR and 3DGRT's code release replaced polygonal cages with oriented bounding boxes (OBBs), which allow significantly faster updates, albeit with less accurate bounding. Although OBBs are not natively supported in OptiX, they can be emulated efficiently through two-level instancing [Wald et al. 2020], where a top-level acceleration structure (TLAS) transforms instanced copies of a bottom-level acceleration structure (BLAS) which consists of a single unit bounding box.

In Tab. 2 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. Because of this, training with OBBs becomes two times faster than icosahedrons under DI (01:22:12 → 40:05), while FPS is slightly lower (145 → 131).

Table 2. BVH update times become critical under DI.

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

Detached Hybrid Transparency (DHT). To avoid evaluating occluded Gaussians, 3DGRT proposed early ray termination: accumulation terminates once transmittance falls below a threshold $\tau_{\min}$, skipping the remaining Gaussians along the ray. While this is safe at runtime (i.e., rendering after optimization has finished), their naive implementation destabilizes training and requires different threshold values at inference ($\tau_{\min} = 0.03$) than during training ($\tau_{\min} = 0.001$). To address this, RaySplats and EPBRR approximate the contribution of the skipped Gaussians to reduce the gradient's bias. We further propose using unordered alpha blending [McGuire and Bavoil 2013; Meshkin 2007] as an estimator and rename this technique Detached Hybrid Transparency [Hahlbohm et al. 2025; Maule et al. 2013] for clarity. DHT stabilizes training under early termination, enabling early ray termination to be used safely during training. This is exemplified in fig. 5 and further explained in the supplemental.

More specifically, during BVH traversal we compute the exact final transmittance τ_{exact} (which can be done out of order). Then, during accumulation, after reaching a transmittance value $\tau < \tau_{\min}$, we approximate the color of the "tail" (the skipped Gaussians) with

$$\boldsymbol{c}_{\text{tail}} = \frac{1}{\alpha_{\text{tail}}} \sum_{i \in G_{\text{tail}}} \alpha_i \boldsymbol{c}_i \,,$$

given $\alpha_{\text{tail}} = \sum_{i \in G_{\text{tail}}} \alpha_i$, where G_{tail} are the skipped Gaussians.[5] The final color $\boldsymbol{c}$ for that pixel becomes

$$\boldsymbol{c} = \boldsymbol{c}_{\text{head}} + \text{sg}\left((\tau - \tau_{\text{exact}})\boldsymbol{c}_{\text{tail}}\right),$$

where $\boldsymbol{c}_{\text{head}}$ is the accumulated color for all sorted Gaussians and $\text{sg}(x)$ denotes the "stop-gradient", operator which detaches this value, treating it as constant during gradient computation.

[5]We define $\boldsymbol{c}_{\text{tail}} = \boldsymbol{0}$ if $\alpha_{\text{tail}} = 0$.

Fig. 5. Test view after training for 1000 iterations with a very high truncation threshold $\tau_{\min} = 0.1$ without (left) and with DHT (center), compared to the ground truth (right).

Tab. 3 shows our method trains stably at high transmittance thresholds ($\tau_{\min}$=0.03) while skipping the accumulation of 40% of hit Gaussians. Interestingly, it reaches slightly better LPIPS than when early termination is disabled ($\tau_{\min} = 0.0$). 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 (GRAY$_{\text{NoDHT}}$) shows rapid deterioration of quality metrics when truncation levels increase.

Table 3. DHT stabilizes early ray termination during training ($\tau_{\min} = 0.03$).

Variant	PSNR$_\uparrow$	LPIPS$_\downarrow$	Opt Time$_\downarrow$	% Skipped
GRAY	**26.47**	**0.236**	05:40	40.28
GRAY$_{\text{NoEarlyTerm.}}$	26.46	0.241	06:33	0.00
GRAY$_{\text{NoDHT}}$	24.54	0.284	**05:34**	54.53

Per-Pixel Linked List Replay Buffers. We found per-pixel linked lists [Yang et al. 2010] (PPLLs) to still be beneficial even under increased Gaussian counts. Because per-pixel Gaussian arrays do not fit in register memory, 3DGRT uses a collection strategy that processes them in fixed-size batches of 16 using multiple BVH traversals to avoid register spilling. Unfortunately, the overhead of performing multiple traversals is substantial. EPBRR instead performs a single BVH traversal and stores all intersected Gaussians in a PPLL replay buffer, deliberately trading memory for speed. The buffer is then replayed to collect batches of Gaussians without additional traversals, improving performance overall. EPBRR also stored temporary data for the backward pass in PPLLs since they allow for coalesced memory access.

We adopt PPLLs for our ray tracer for both the forward and backward pass, but we found EBPRR's implementation suboptimal, as too many attributes were stored in lists even when they can be recomputed efficiently. Our implementation uses lightweight PPLLs which only store Gaussian IDs, hit distances, and alpha values.

We validated that PPLLs are still worth using since DI reduces the required BVH traversals when using 3DGRT's Gaussian collection strategy. Tab. 4 shows that PPLLs are still much faster even in the DI regime, increasing frame rates by around 40% (175 → 248). [6]

Weight-Based Pruning. We propose changing the pruning approach to further cut down on bloated Gaussian counts. Several recent approaches, including EPBRR, explore pruning unimportant Gaussians [Fang and Wang 2024; Girish et al. 2024; Hanson et al. 2025a,b; Niemeyer et al. 2025;

[6]Note that we disabled DHT for the multiple traversals used in 3DGRT since the list of unprocessed Gaussians isn't readily available; implementing DHT for it would require an additional traversal, further decreasing speed.

Table 4. PPLLs still outmatch multiple traversals under DI.

Variant	Collection Method	FPS↑
GRay$_{\text{MultipleTraversal}}$	Multiple BVH Traversals	175
GRay	Per-Pixel Linked Lists	**248**

Papantonakis et al. 2024; Poirier-Ginter et al. 2025; Sabour et al. 2024]. Inspired by these ideas, we adopt a simple weight-based pruning criterion, that is to prune according to the average accumulated weight of a Gaussian over views that observe it:

$$\left[\frac{1}{|V_i|\,|v|}\sum_{v\in V_i}\sum_{r\in v} w_i(r)\right] < \gamma_{\text{prune}}\,.$$

Here, v is a training view from the set of views V_i that see Gaussian i, r is a ray in that view, and $w_i(r)$ is the Gaussian's accumulation weight for that ray. This is done every 500 iterations.

Tab. 5 shows the impact our of pruning method. Our selected value value $\gamma_{\text{prune}} = 10^{-7}$ reduces the Gaussian count to less than half (3.7M → 1.5M) while *improving* PSNR, and only worsening LPIPS by small amounts (0.234 → 0.236). This results in substantial improvement to FPS (189 → 248) and to optimization times (7:26 → 5:40). Our weight-based pruning significantly outperforms opacity-based pruning, which we show in the supplemental with extensive parameter sweeps.

Table 5. Aggressive weight-based pruning maintains good quality ($\gamma_{\text{prune}} = 10^{-7}$).

Variant	PSNR↑	LPIPS↓	Opt. Time↓	FPS↑	Final #Gaussians↓
GRay	**26.47**	0.236	**05:40**	**248**	**1.52M**
GRay$_{\text{NoPruning}}$	26.42	**0.234**	07:26	189	3.27M

Halving Training Iterations. One of EDGS's main claims is that DI converges more rapidly than SI [Kotovenko et al. 2025]. Thus, benefiting from DI's faster convergence, we train for 15K iterations instead of 30K. Of course, training at reduced iteration counts can lead to reduced quality. To compensate, we adjust learning rates by adding schedules for scale, rotation, and SH DC coefficients. These schedules assign higher values early during training while decaying them down progressively to standard values. We also increase the SH higher-order coefficient learning rate 5× to 0.000625. The exact schedules used are described in the supplemental.

Tab. 6 shows that it is in fact possible to train at 15K iterations with slightly better LPIPS score than 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. In the supplemental, additional configurations are presented and the learning rate schedules are ablated.

Table 6. Fewer iterations can still converge fully.

Variant	Iterations	PSNR↑	LPIPS↓	Opt. Time↓
GRay$_{\text{30K}}$	30000	**26.51**	0.243	10:24
GRay	15000	26.47	**0.236**	**05:40**

Scale Decay. To further encourage smaller Gaussians, we included a scale decay. At each iteration, Gaussians are scaled down by a factor of $\eta_{\text{decay}} = 0.999875$. We found this regularizer highly effective at controlling final Gaussian sizes while also encouraging the pruning of occluded or out-of-view Gaussians, which increases final FPS by noticeable amounts while only impacting quality minimally.

Tab. 7 shows that increasing decay improves FPS by large amounts (157 $\rightarrow$ 248), which is explained by both reduced final Gaussian average scales (0.0211 $\rightarrow$ 0.0166) and an increase in pruning (1.73M $\rightarrow$ 1.52M final Gaussian count). Since decay also reduces quality slightly, we conclude that scale decay is a highly effective lever for balancing quality and speed. An extensive sweep showing this is provided in the supplemental.

Table 7. Scale decay efficiently trades quality for speed ($\eta_{\text{decay}} = 999875$).

Variant	PSNR$_\uparrow$	LPIPS$_\downarrow$	FPS$_\uparrow$	Final #Gaussians$_\downarrow$	Final Avg Scale
GRay	26.47	0.236	**248**	**1.52M**	**0.0166**
GRay$_{\text{NoDecay}}$	**26.60**	**0.231**	157	1.73M	0.0211

Initialization Binning. DI densely covers surfaces with Gaussians, but can result in highly redundant Gaussians when the selected reference views are similar. We propose using binning to remove duplicate Gaussians at initialization. In practice, we voxelize all Gaussians positions at a fixed grid of size $\psi_{\text{bin}} r_{\text{scene}}$, where $\psi_{\text{bin}} = 0.04$ is the grid cell size and the scene radius r_{scene} is defined as 1.1 times the maximum distance from the average camera center to any other camera center [Zhang et al. 2020][7]. The position, color and scale properties of each Gaussians within a voxel are averaged.

Table 8. Many initial DI Gaussians are useless ($\psi_{\text{bin}} = 0.04$).

Variant	PSNR$_\uparrow$	LPIPS$_\downarrow$	Init #Gaussians$_\downarrow$
GRay	26.47	0.236	**3.27M**
GRay$_{\text{NoBinning}}$	**26.49**	**0.234**	3.97M

Our experiments show that this practice maintains visual quality nearly identical while reducing initial Gaussian counts significantly. Tab. 8 gives our final setting $\psi_{\text{bin}} = 0.04$, which removes over 15% of the initial Gaussians at limited cost to PSNR (26.49 $\rightarrow$ 26.47) and LPIPS (0.238 $\rightarrow$ 0.239). An parameter sweep is also provided in the supplemental.

Additional Changes. Unlike 3DGS and 3DGRT, we do not use opacity resets, and unlike 3DGS, do not prune Gaussians based on size. Remaining small performance differences may stem from implementation improvements. All differences are detailed in the supplemental.

6 Experiments

In this subsection, we compare our ray tracer GRay to 3DGRT and 3DGS under various configurations, and also run EDGS's original method for reference.

More specifically, we extracted the dense initialization code from EDGS's codebase and incorporated it into 3DGS, 3DGRT, and our ray tracer GRay while dispensing of any other changes.[8] We

[7] A single value $\psi_{\text{bin}} = 0.04$ worked well in the 3DGS benchmark scenes but might require adjustment for larger scenes.
[8] The original EDGS implementation includes other minor changes like an opacity decay and different LR schedule, which improve quality slightly but make precise comparisons harder. For this reason we exclude such changes and apply DI directly to 3DGS. This is further discussed in the supplemental.

Table 9. Reconstruction quality and performance between methods and configurations.

Method	PSNR↑	SSIM↑	LPIPS↓	Init Time↓	Opt Time↓	FPS↑	Init #G↓	Final #G↓	#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
$3DGS_{SI+15Kiters}$	26.55	0.814	0.296	00:00	02:47	299	0.11M	1.87M	15000
$3DGS_{DI+15Kiters}$	26.68	0.830	0.231	01:58	05:37	235	3.97M	3.39M	15000
$3DGS_{SI+RegularAdam}$	27.28	0.836	0.253	00:00	18:13	190	0.11M	2.49M	30000
$3DGS_{DI+RegularAdam}$	27.29	0.838	0.211	01:58	23:56	157	3.97M	3.50M	30000
$EDGS_{RegularAdam}$	27.55	0.850	0.204	00:00	30:38	173	3.98M	2.26M	30000
$EDGS_{RegularAdam+15Kiters}$	27.26	0.851	0.212	00:00	20:07	167	3.98M	2.16M	15000
RayGaussX	28.14	0.856	0.221	00:00	56:18	39	0.11M	3.28M	30000
$3DGRT_{SI}$	26.77	0.828	0.258	00:00	55:01	68	0.11M	3.24M	30000
$3DGRT_{DI}$	26.46	0.824	0.229	01:58	40:05	131	3.97M	2.64M	30000
$3DGRT_{SI+15Kiters}$	26.05	0.818	0.268	00:00	24:04	50	0.11M	3.39M	15000
$3DGRT_{DI+15Kiters}$	26.28	0.830	0.230	01:58	20:13	120	3.97M	2.62M	15000
$3DGRT_{SI+SparseAdam}$	26.61	0.825	0.264	00:00	53:37	52	0.11M	3.32M	30000
$3DGRT_{DI+SparseAdam}$	26.16	0.815	0.239	01:58	33:43	136	3.97M	2.69M	30000
GRay	26.47	0.819	0.236	01:58	05:40	248	3.27M	1.52M	15000

used the indoor RoMA model for all indoor scenes and the outdoor one for all outdoor scenes. For simplicity we ran dense initialization on the full-resolution images, but worked with downsized images, as was done in the 3DGS paper. We also compared to RayGaussX, a slower high quality ray tracing method. We standardized downsizing across all methods, using Pillow's Lanczos filter. We reported averages over all thirteen Mip-NeRF360, Tanks & Temples, and Deep Blending scenes which are the standard 3DGS benchmark introduced in the original paper [Kerbl et al. 2023].

For 3DGS, we used the improved optimizer proposed by Taming 3DGS [Mallick et al. 2024]; for 3DGRT, we did not use Sparse Adam in most experiments, since it reduced quality slightly without providing a significant performance boost. We ran 3DGRT with oriented bounding boxes unless otherwise specified. We measured FPS over all test views with a warm start[9] and averaged their values directly. We also reported PSNR, SSIM [Wang et al. 2004], and LPIPS [Zhang et al. 2018] as usual. Note that the original 3DGS implementation computed slightly erroneous LPIPS scores which are unfortunately reported in numerous works [Rota Bulò et al. 2024]; we reported accurate LPIPS for all methods.

Rendered videos and captured interactive sequences are also provided alongside the paper. Unfortunately, dense initialization seems prone to floaters, especially in outdoor scenes with vegetation—this applies to all methods tested with DI. As such, for all qualitative results we masked floaters with manually configured near clipping planes with distances indicated in the supplemental. The same clipping planes were used for all methods; they were not used when measuring quality.

Tab. 9 contains our main comparisons; 3DGS was run with Taming 3DGS's techniques unless marked with $\cdot_{RegularAdam}$ while 3DGRT was run with the regular optimizer unless marked with $\cdot_{SparseAdam}$. We ran all experiments on a single NVIDIA GeForce RTX 4090.

GRay Runs as Fast as 3DGS Near 3DGRT's Quality. Our main results are shown directly in Fig. 1: GRay roughly matches the optimization times and FPS of 3DGS while maintaining a quality level close to 3DGRT (worse PSNR and SSIM, but better LPIPS). We credit the majority of this improvement to DI. Note, however, that DI takes roughly two minutes to run, and so the total training times (initialization + optimization) remain in favor of 3DGS, along with the overall visual

[9]To ensure caches are filled, we run the FPS measurement twice and retain the second result.

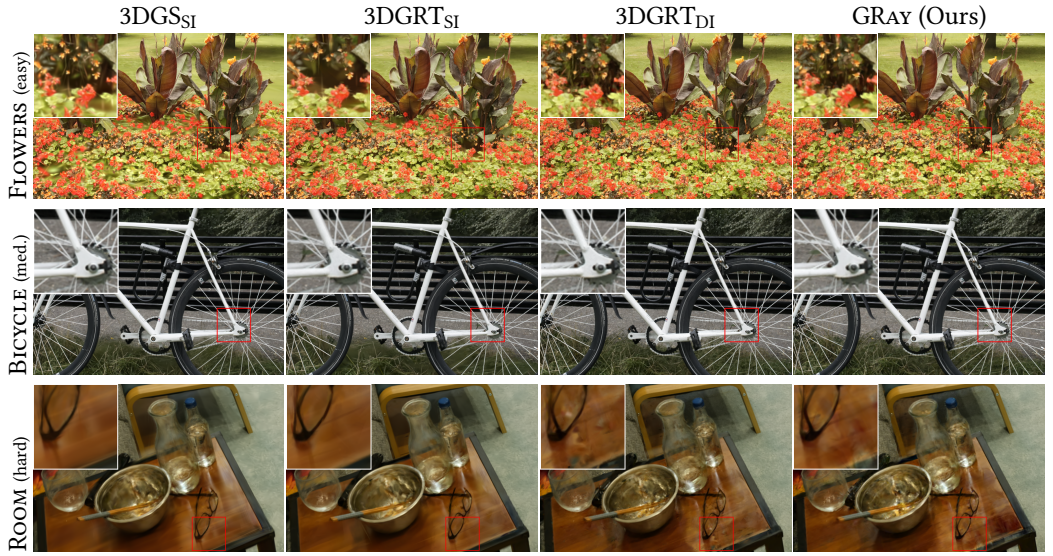

Fig. 6. Novel view rendering on FLOWERS (an easy case for dense initialization), BICYCLE (an average case), and ROOM (a hard case for dense initialization).

quality. FPS are also nearly 4× faster for our method compared to 3DGRT (68 → 248). Finally, despite starting at highly inflated Gaussian counts, our weight-based pruning results in lower final counts (3.24M → 1.52M) which ultimately saves memory.

Splatting Fails to Exploit DI for Speed. Looking at overall optimization times and frame rates, we can see that optimization speed (on a per-iteration basis) is consistently worse for 3DGS under DI (06:18 → 10:09), and that the frame rates after optimization completes are also slightly worse (253 → 241). In fact, optimization times after halving iteration count with DI do not even improve much over regular training (3DGS$_{DI+15Kiters}$@05:37 ≈ 3DGS$_{SI}$@06:18). In great contrast, ray tracing with 3DGRT obtains a marked 2–3× FPS increase when switching to DI in all tested configurations, with improved optimization times as well (55:01 → 40:05 in the main configuration). We conclude that splatting fails to leverage DI for speed. This means our ray tracer's speed remains comparable to 3DGS even when the latter is also switched to DI.

DI can Lower PSNR/SSIM but Improves LPIPS. While the original EDGS method improved PSNR and SSIM over 3DGS, they also included a modified learning rate schedule and opacity decay which improved visual quality irrespective of DI. When switching 3DGS or 3DGRT to DI with no other changes applied, we found that PSNR and SSIM tend to worsen slightly (most notably PSNR 26.77 → 26.46 for 3DGRT), while LPIPS improved considerably (for 3DGS 0.262 → 0.226; for 3DGRT 0.258 → 0.229). While LPIPS is arguably a better assessment of human perceived perceptual quality [Blau and Michaeli 2018; Ding et al. 2021; Zhang et al. 2018], in practice DI performs better in certain types of scenes and worse in others, which we discuss below.

7 Discussion

We showed that ray tracing can be greatly accelerated by exploiting its synergy with dense initialization, which we attribute to its advantage in computational scaling over splatting. We discuss below the implications and main limitations of our work.

DI's Quality Profile. Most of the quality difference between our method and 3DGRT is explained by the switch to DI. DI exhibits a notably different quality profile: it appears much better at reconstructing high-frequency diffuse details like vegetation (as already discussed in EDGS), but appears notably worse at reconstructing highly reflective, smooth objects. We hypothesize that this is because DI struggles to initialize Gaussians at their correct location when objects are highly reflective, and/or because larger Gaussians tend to smooth out details which leaves fewer artifacts. Fig. 6 shows such cases for all three methods.

Performance vs. Splatting. While our method is significantly faster than 3DGRT and near its quality, splatting continues to outperform ray tracing, achieving better reconstruction quality at comparable frame rates, or higher frame rates at equal quality. For these reasons, rasterization is unlikely to be fully replaced. Nonetheless, ray tracing proves highly effective in the dense initialization regime: our results demonstrate that the performance gap between splatting and ray tracing can be closed. Moreover, ray tracing naturally supports light-transport simulation; producing robust, high-quality, relightable 3D Gaussian scenes remains an open challenge, for which fast ray tracing is likely to become a crucial enabling tool.

Moreover, our results suggest that ray tracing can scale more efficiently to high Gaussian densities relative to the pixel density, while splatting may scale better to higher resolution unless more Gaussians are added. To test this, we re-rendered all trained scenes at different resolutions, by scaling the existing resolution up and down by a constant factor. We start at the resolutions from the original 3DGS evaluation, which average around 1.2 megapixels. Results are shown in fig. 7: as expected, our ray tracing method appears slower than 3DGS at larger resolutions, but several times faster at smaller resolutions when the same Gaussians are rendered. Interestingly, splatting slows down significantly such low resolutions.[10]

Compatibility with Splatting. Ray tracing also addresses two key limitations of 3D Gaussian Splatting: popping artifacts and the affine approximation. Popping occurs because 3DGS sorts Gaussians per primitive rather than per pixel; as the camera or scene changes, small variations in visibility ordering can lead to abrupt changes. By contrast, ray tracing naturally sorts Gaussians along each viewing ray, ensuring a consistent compositing order that eliminates such pops. The affine approximation, on the other hand, arises from the projection model used in 3DGS, where each Gaussian is approximated by an affine transform in screen space. This assumption breaks under strong perspective effects, leading to visible distortion. Ray tracing resolves this issue entirely by evaluating each Gaussian exactly in 3D under perspective projection.

While these differences mean the two rendering paradigms are not directly compatible, prior work has demonstrated promising solutions. In particular, Halbohm et al. [Hahlbohm et al. 2025] introduced a rasterizer capable of evaluating Gaussian kernels exactly without relying on the affine approximation, while also mitigating popping through hybrid transparency.[11] Similarly, *Stop the Pop* [Radl et al. 2024] showed that the sorting discrepancy between splatting and ray tracing can be reconciled. Finally, if popping is tolerated, ray tracing can emulate per-primitive sorting without issue. These insights suggest that it should be possible to design a compatible rasterizer–ray tracer pair, unifying both under a shared framework, even though such an implementation does not yet exist.

[10]Do note however that ray tracing only intersects the Gaussians at the exact center of each pixel; at very small resolutions Gaussians can fall in-between pixels centers and be skipped, which can explain the performance difference beyond algorithmic complexity.

[11]Their hybrid-transparency method includes gradients and truncates at a fixed primitive count, whereas ours truncates at a specific transmittance threshold, but the ideas are otherwise very similar.

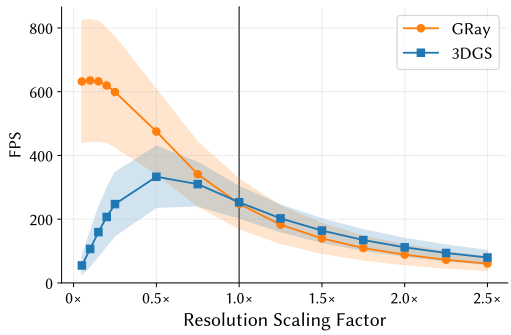

Fig. 7. Frame rates at different resolutions measured after training, for GRay and 3DGS. We rescaled all scene resolutions by a constant scaling factor and report averages over all scenes, with shaded regions indicating ± one standard deviation.

Limitation: Higher Memory Use. Our method uses more memory than 3DGRT, as a consequence from choosing PPLLs. To verify this, we profiled the Bicycle scene and measured its peak memory consumption. Our method peaks at 16.52 GB of VRAM, whereas 3DGRT only peaks at 12.42 GB. For reference, 3DGS peaks at 14.02 GB with SI, and 11.62 GB with DI. All scenes ran successfully with roughly 24 GB of VRAM, which corresponds to the recommended minimum value in 3DGS's official implementation.

Here is a breakdown of our memory consumption: we ran all scenes with a preallocated forward pass PPLL of size 300M (which uses over 4.8 GB of VRAM) and a preallocated backward PPLL of size 120M (which uses over 1.9 GB of VRAM). We additionally store 1005 bytes of data per Gaussian; Gaussians take 3.62 GB of VRAM at the start of training for the Bicycle scene, which decreases to 2.23 GB at the end of training due to pruning. We also store all test images in VRAM, which requires 2.36 GB for this scene. In total, 14.5 GB is allocated before training even starts.

Scaling training to higher resolutions would require very large PPLLs, which would take up considerably more memory (linear in the number of pixels). However, this usage could be reduced by rendering tiles instead of the whole image at once.

8 Conclusion

Gaussian ray tracing has several advantages, including avoiding the affine approximation and popping artifacts, and is of critical importance for inverse rendering and relighting applications. As of yet however, it has remained significantly slower than splatting. We argue that ray tracing can better handle increasing counts of tiny Gaussians, where, unlike splatting, it can exhibit logarithmic scaling. Our experiments on dense initialization suggest that this effect is of great practical significance. In addition, we have analyzed several important algorithmic components and parameters of Gaussian ray tracing for radiance fields. Of these, per-pixel linked lists and scale decay have the most significant effect on performance. Leveraging our analysis, we propose novel ray tracing solution designed to exploit ray tracing's algorithmic scaling and dense initialization. Our ray tracer closes the gap in performance between 3DGRT and 3DGS while maintaining a quality level near 3DGRT's. While quality remains somewhat lower than 3DGS, the significant increase in speed will be a key enabler for future work, especially in cases where light transport simulation is required. Our code will be open sourced on publication.

Acknowledgments

Thanks to Jeffrey Hu for pointing us towards dense initialization and programming help, and to Ishaan Shah for his work on the Gaussian Viewer. This research was co-funded by the European Union (EU) ERC Advanced Grant NERPHYS No 101141721 (see https://project.inria.fr/nerphys). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the EU or the European Research Council. Neither the EU nor the granting authority can be held responsible for them. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). This research was also supported by NSERC grant RGPIN-2020-04799 and the Digital Research Alliance Canada. The authors are grateful to Adobe and NVIDIA for generous donations.

References

Hugo Blanc, Jean-Emmanuel Deschaud, and Alexis Paljic. 2025a. Raygauss: Volumetric gaussian-based ray casting for photorealistic novel view synthesis. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 1808–1817.

Hugo Blanc, Jean-Emmanuel Deschaud, and Alexis Paljic. 2025b. RayGaussX: Accelerating Gaussian-Based Ray Marching for Real-Time and High-Quality Novel View Synthesis. arXiv:2509.07782 [cs.CV] https://arxiv.org/abs/2509.07782

Yochai Blau and Tomer Michaeli. 2018. The perception-distortion tradeoff. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6228–6237.

Krzysztof Byrski, Marcin Mazur, Jacek Tabor, Tadeusz Dziarmaga, Marcin Kądziołka, Dawid Baran, and Przemysław Spurek. 2025a. Raysplats: Ray tracing based gaussian splatting. *arXiv preprint arXiv:2501.19196* (2025).

Krzysztof Byrski, Grzegorz Wilczyński, Weronika Smolak-Dyżewska, Piotr Borycki, Dawid Baran, Sławomir Tadeja, and Przemysław Spurek. 2025b. REdiSplats: Ray Tracing for Editable Gaussian Splatting. *arXiv preprint arXiv:2503.12284* (2025).

Guikun Chen and Wenguan Wang. 2024. A Survey on 3D Gaussian Splatting. *ArXiv* abs/2401.03890 (2024). https://api.semanticscholar.org/CorpusID:266844057

Yun Chen, Matthew Haines, Jingkang Wang, Krzysztof Baron-Lis, Sivabalan Manivasagam, Ze Yang, and Raquel Urtasun. 2025. SaLF: Sparse Local Fields for Multi-Sensor Rendering in Real-Time. *arXiv preprint arXiv:2507.18713* (2025).

Jorge Condor, Sebastien Speierer, Lukas Bode, Aljaz Bozic, Simon Green, Piotr Didyk, and Adrian Jarabo. 2025. Don't Splat your Gaussians: Volumetric Ray-Traced Primitives for Modeling and Rendering Scattering and Emissive Media. *ACM Transactions on Graphics* 44, 1 (2025), 1–17.

Keyan Ding, Kede Ma, Shiqi Wang, and Eero P Simoncelli. 2021. Comparison of full-reference image quality models for optimization of image processing systems. *International Journal of Computer Vision* 129, 4 (2021), 1258–1281.

Johan Edstedt, Qiyu Sun, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. 2024. RoMa: Robust Dense Feature Matching. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Guangchi Fang and Bing Wang. 2024. Mini-splatting: Representing scenes with a constrained number of gaussians. In *European Conference on Computer Vision*. Springer, 165–181.

Jian Gao, Chun Gu, Youtian Lin, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. 2023. Relightable 3D Gaussian: Real-time Point Cloud Relighting with BRDF Decomposition and Ray Tracing. *arXiv:2311.16043* (2023).

Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. 2024. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *European Conference on Computer Vision*. Springer, 54–71.

Shrisudhan Govindarajan, Daniel Rebain, Kwang Moo Yi, and Andrea Tagliasacchi. 2025. Radiant Foam: Real-Time Differentiable Ray Tracing. *arXiv:2502.01157* (2025).

Chun Gu, Xiaofei Wei, Zixuan Zeng, Yuxuan Yao, and Li Zhang. 2025. Irgs: Inter-reflective gaussian splatting with 2d gaussian ray tracing. In *Proceedings of the Computer Vision and Pattern Recognition Conference*. 10943–10952.

Florian Hahlbohm, Fabian Friederichs, Tim Weyrich, Linus Franke, Moritz Kappel, Susana Castillo, Marc Stamminger, Martin Eisemann, and Marcus Magnor. 2025. Efficient Perspective-Correct 3D Gaussian Splatting Using Hybrid Transparency. In *Computer Graphics Forum*. Wiley Online Library, e70014.

Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. 2025a. Speedy-Splat: Fast 3D Gaussian Splatting with Sparse Pixels and Sparse Primitives. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*. 21537–21546. https://speedysplat.github.io/

Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. 2025b. PUP 3D-GS: Principled Uncertainty Pruning for 3D Gaussian Splatting. In *Proceedings of the Computer Vision and Pattern Recognition*

Conference (CVPR). 5949–5958. https://pup3dgs.github.io/

Bernhard Kerbl. 2025. The Impact and Outlook of 3D Gaussian Splatting. *arXiv preprint arXiv:2510.26694* (2025).

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Dmytro Kotovenko, Olga Grebenkova, and Bjorn Ommer. 2025. EDGS: Eliminating Densification for Efficient Convergence of 3DGS. *ArXiv* abs/2504.13204 (2025). https://api.semanticscholar.org/CorpusID:277940156

Junseo Lee, Sangyun Jeon, Jungi Lee, Junyong Park, and Jaewoong Sim. 2026. GRTX: Efficient Ray Tracing for 3D Gaussian-Based Rendering. *arXiv preprint arXiv:2601.20429* (2026).

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

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

Marilena Maule, João Comba, Rafael Torchelsen, and Rui Bastos. 2013. Hybrid transparency. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Orlando, Florida) *(I3D '13)*. Association for Computing Machinery, New York, NY, USA, 103–118. doi:10.1145/2448196.2448212

Morgan McGuire and Louis Bavoil. 2013. Weighted Blended Order-Independent Transparency. *Journal of Computer Graphics Techniques (JCGT)* 2, 2 (18 December 2013), 122–141. http://jcgt.org/published/0002/02/09/

Houman Meshkin. 2007. Sort-Independent Alpha Blending. GDC Vault — GDC 2007 Session. https://www.gdcvault.com/play/535/Sort-Independent-Alpha Perpetual Entertainment.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 2024. 3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes. *ACM Transactions on Graphics and SIGGRAPH Asia* (2024).

Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. 2025. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. In *2025 International Conference on 3D Vision (3DV)*. IEEE, 134–144.

Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. 2024. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7, 1 (2024), 1–17.

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics* (August 2010).

Yohan Poirier-Ginter, Jeffrey Hu, Jean-François Lalonde, and George Drettakis. 2025. Editable Physically-based Reflections in Raytraced Gaussian Radiance Fields. In *SIGGRAPH Asia 2025 - 18th ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia*. Hong Kong, Hong Kong SAR China. doi:10.1145/3757377.3763971

Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. 2024. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. *ACM Transactions on Graphics* 4, 43, Article 64 (2024).

Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. 2024. Revising densification in gaussian splatting. In *European Conference on Computer Vision*. Springer, 347–362.

Sara Sabour, Lily Goli, George Kopanas, Mark Matthews, Dmitry Lagun, Leonidas Guibas, Alec Jacobson, David J. Fleet, and Andrea Tagliasacchi. 2024. SpotLessSplats: Ignoring Distractors in 3D Gaussian Splatting. *arXiv:2406.20055* (2024).

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Michael Steiner, Thomas Köhler, Lukas Radl, Felix Windisch, Dieter Schmalstieg, and Markus Steinberger. 2025. AAA-Gaussians: Anti-Aliased and Artifact-Free 3D Gaussian Rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 27650–27659.

Xin Sun, Iliyan Georgiev, Yun Fei, and Miloš Hašan. 2025. Stochastic Ray Tracing of 3D Transparent Gaussians. *arXiv preprint arXiv:2504.06598* (2025).

Shimon Ullman. 1979. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203, 1153 (1979), 405–426.

Ingo Wald, Nate Morrical, Stefan Zellmann, Lei Ma, Will Usher, Tiejun Huang, and Valerio Pascucci. 2020. Using Hardware Ray Transforms to Accelerate Ray/Primitive Intersections for Long, Thin Primitive Types. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 2, Article 17 (Aug. 2020), 16 pages. doi:10.1145/3406179

Xinzhe Wang, Ran Yi, and Lizhuang Ma. 2024. Adr-gaussian: Accelerating gaussian splatting with adaptive radius. In *SIGGRAPH Asia 2024 Conference Papers*. 1–10.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612. doi:10.1109/TIP.2003.819861

Qi Wu, Janick Martinez Esturo, Ashkan Mirzaei, Nicolas Moenne-Loccoz, and Zan Gojcic. 2025. 3DGUT: Enabling Distorted Cameras and Secondary Rays in Gaussian Splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)* (2025).

Tao Xie, Xi Chen, Zhen Xu, Yiman Xie, Yudong Jin, Yujun Shen, Sida Peng, Hujun Bao, and Xiaowei Zhou. 2024. EnvGS: Modeling View-Dependent Appearance with Environment Gaussian. *arXiv preprint arXiv:2412.15215* (2024).

Jason C. Yang, Justin Hensley, Holger Gruen, and Nicolas Thibieroz. 2010. Real-Time Concurrent Linked List Construction on the GPU. *Computer Graphics Forum* (2010). doi:10.1111/j.1467-8659.2010.01725.x

Yuxuan Yao, Zixuan Zeng, Chun Gu, Xiatian Zhu, and Li Zhang. 2024. Reflective gaussian splatting. *arXiv preprint arXiv:2412.19282* (2024).

Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492* (2020).

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.