

Accelerating 3D Gaussian Ray Tracing with Ellipsoid-Based Kd-trees

Junsoo Kim
Sogang University
Seoul, Korea
goliath12@sogang.ac.kr

Insung Ihm
Sogang University
Seoul, Korea
ihm@sogang.ac.kr

Abstract

The Bounding Volume Hierarchy (BVH) is widely used as a spatial acceleration structure for 3D Gaussian ray tracing, primarily because it directly leverages the dedicated ray tracing hardware of modern GPUs. However, unlike conventional surface rendering, it inherently imposes a significant computational bottleneck during the volume rendering of Gaussian splats, where intersections along a ray must be strictly sorted by depth. In this work, we propose using an ellipsoid-based kd-tree structure that naturally enumerates these intersections in order, thereby enabling early ray termination without exhaustive tree traversal. Through extensive evaluations, we demonstrate the effectiveness of the new data structure in accelerating 3D Gaussian ray tracing. Notably, its independence from hardware ray tracing units allows for an efficient parallel implementation on resource-constrained mobile platforms.

ACM Reference Format:

Junsoo Kim and Insung Ihm. 2026. Accelerating 3D Gaussian Ray Tracing with Ellipsoid-Based Kd-trees. In *SIGGRAPH Asia 2026 Posters (SA Posters '26)*, December 01–04, 2026, Kuala Lumpur, Malaysia. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3829333.3847910>

1 Background

While computationally more expensive than rasterization-based rendering approaches [Kerbl et al. 2023], ray tracing was also explored to visualize 3D Gaussian splats for its inherent capacity to accurately simulate advanced rendering effects and camera models. For efficient ray tracing, the method presented in [Moenne-Loccoz et al. 2024] approximates semi-transparent Gaussian splats as ellipsoids, representing each with a bounding triangular mesh and organizing them into a Bounding Volume Hierarchy (BVH). By encapsulating 3D Gaussian splats within a BVH structure, it directly leverages modern GPU ray tracing hardware to achieve real-time rendering performance on PC platforms.

The core of 3D Gaussian ray tracing lies in its volume rendering process, where rays are traced to accumulate color and opacity in a front-to-back manner until the accumulated opacity reaches a predefined threshold, triggering early ray termination. To effectively implement this mechanism, it is critical to efficiently sort the Gaussian splats by increasing depth along each ray. In [Moenne-Loccoz et al. 2024], an effective volume rendering algorithm was

described within the NVIDIA OptiX programming model: a *ray-gen* program traverses the BVH and, combined with an *any-hit* program, constructs a sorted array of the k front-most Gaussian splats along a ray to accumulate them in order. This ray-gen program is invoked repeatedly to process subsequent batches of k splats until all intersections are evaluated or early ray termination is triggered. Although reported as optimal for dedicated GPU ray tracing pipelines, this method necessitates multiple BVH traversals to isolate only the front-most splats that actively contribute to the volume rendering, thereby introducing substantial inefficiency.

This inefficiency is mainly driven by the inherent structural property of the BVH, where the bounding volumes of child nodes within an internal node often overlap. This overlap frequently necessitates traversing both children to determine the nearest intersection. While this does not introduce significant overhead in conventional surface rendering, it creates a substantial computational bottleneck in volume rendering, which requires repeatedly finding the next nearest intersections. Moreover, the strict reliance on frequent any-hit program invocations significantly degrades the performance of each ray-gen execution. In particular, the hardware-assisted BVH-based volume rendering becomes a critical bottleneck on mobile GPUs, where computational resources are relatively constrained.

2 Our contribution

To overcome the inherent limitations of the BVH-based framework in 3D Gaussian ray tracing, we propose adopting kd-tree spatial subdivision for efficient volume rendering. By partitioning space without overlapping boundaries, kd-trees not only effectively accelerate ray-scene intersections but also naturally support an efficient front-to-back traversal. Initially, we construct a kd-tree using a standard algorithm [Wald and Havran 2006] over the proxy triangular meshes, where each Gaussian splat is represented by 20 triangles. However, this often results in an inefficient, large tree with high triangle redundancy in the leaf nodes, as triangles that intersect partitioning planes are repeatedly duplicated across child nodes during construction. Therefore, in the next step, we optimize the leaf nodes by collapsing the duplicated triangles of each Gaussian splat back into its original 3D ellipsoid representation, parameterized by a center $\mu \in \mathbb{R}^3$ and a covariance matrix $\Sigma \in \mathbb{R}^{3 \times 3}$.

Transitioning to this ellipsoid-based hierarchy necessitates key modifications to the standard construction algorithm. Most notably, the Surface Area Heuristic (SAH) cost function used to determine optimal partitioning planes must incorporate the number of unique ellipsoids, N , within an AABB rather than the raw triangle count, because primitive intersection tests at the leaf nodes are executed per ellipsoid. Furthermore, unlike conventional surface rendering, the evaluation cost of a leaf node is no longer



This work is licensed under a Creative Commons Attribution 4.0 International License. *SA Posters '26, Kuala Lumpur, Malaysia*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2839-6/2026/12
<https://doi.org/10.1145/3829333.3847910>

linearly proportional to its primitive count, as all valid intersections within a visited leaf node must be explicitly depth-sorted prior to opacity accumulation. Given that sorting costs grow non-linearly with the number of intersections, it is crucial to minimize an uneven distribution of primitives across child nodes. To enforce this heuristic, the new cost function combines the standard cost function $C_{std}(V)$ [Wald and Havran 2006] with a new penalty factor: $C_{new}(V) = C_{std}(V) \left(1 + W_{bal} \frac{|N_L - N_R|}{N_L + N_R}\right)$, where $C_{std}(V) = K_T + K_I \left(N_L \frac{SA(V_L)}{SA(V)} + N_R \frac{SA(V_R)}{SA(V)}\right) \cdot B_{empty}$, with W_{bal} being the balancing coefficient and B_{empty} denoting the empty bonus factor.

Finally, to maximize computational efficiency during ray-ellipsoid intersections, we directly compute the sample point along the ray that minimizes the Mahalanobis distance to the Gaussian, thereby evaluating its peak response for opacity accumulation. Consequently, we eliminate the computational overhead of solving a quadratic equation without any perceptible loss in rendering quality.

3 Results

In this study, we conducted experiments across both PC and mobile platforms. On a PC equipped with an NVIDIA GeForce RTX 4090 GPU, we evaluated our proposed CUDA-based kd-tree renderer (pc-kdt). As a baseline representing the BVH-based 3DGRT method [Moenne-Loccoz et al. 2024], we adopted an open-source renderer that implements it within a Vulkan environment [Jean-Eudes et al. 2026] (pc-bvh). Because these PC renderers were not directly portable to a mobile environment, we developed a dedicated OpenCL-based kd-tree renderer (mo-kdt) alongside a cross-platform Vulkan-based BVH renderer (mo-bvh). Due to device compatibility limitations, the mobile evaluations were performed on two devices: the kd-tree on a Samsung Galaxy S26 Ultra smartphone (using a Qualcomm Adreno 840 GPU lacking a dedicated RT pipeline) and the BVH on a Galaxy S26 (using a Samsung Xclipse 960 GPU featuring a dedicated RT pipeline). Even considering the Ultra model’s GPU advantage (by a margin of at most 10%), our experiments clearly demonstrate the performance improvement achieved by our new algorithm. Unlike the BVH baselines that depend on dedicated RT cores, our kd-tree renderers do not require specialized ray tracing hardware.

In our evaluation, the NeRF Synthetic dataset features background-free scenes centered on a single object with a relatively small number of Gaussians. In contrast, the Mip-NeRF 360 dataset consists of high-resolution real-world scenes with omnidirectional backgrounds and a substantially larger number of Gaussian primitives. We empirically set $W_{bal} = 0.5$, $B_{empty} = 0.9$, while the threshold N_{leaf}^{max} limiting the maximum number of ellipsoids per leaf node during tree construction was set to 64 for NeRF Synthetic and 256 for Mip-NeRF 360 (Table 1).

Table 1 shows that our method achieves a frame rate improvement ranging from 1.23× to 3.62× on the PC platform across all scenes except for bicycle. Although the hardware-assisted ray-triangle intersection in the BVH method is computationally cheaper than the analytic ray-ellipsoid intersection in our kd-tree method, the substantial reduction in intersection frequency achieved by our method, as indicated in Fig. 1, outweighs this overhead, leading to a significant overall speedup. As a notable exception, for the bicycle

Table 1: Temporal and spatial performance. Rendering times for selected views are first compared in terms of frame rate between the BVH (bvh) and our kd-tree (kdt) methods, followed by an evaluation of the memory footprint required for the respective tree structures. On the mobile device, the BVH-based method exceeded a three-second execution time, resulting in a rendering failure. The first six and the last six scenes were drawn from the NeRF Synthetic [Mildenhall et al. 2021] and Mip-NeRF 360 [Barron et al. 2022] datasets, respectively. All scenes were pre-trained via the 3DGRT method, and rendered at an FHD+ resolution (2340 × 1080).

Scene (Gaussian #)	Speed (fps)				Memory (MB)	
	pc-bvh	pc-kdt	mo-bvh	mo-kdt	bvh	kdt
hotdog (126K)	139.2	437.4	2.25	19.59	58.9	227.5
mic (155K)	195.2	707.7	3.53	32.05	72.4	39.5
ship (251K)	158.3	270.4	2.29	11.67	117.0	25.6
lego (327K)	192.2	431.8	3.13	19.06	141.0	54.7
drums (348K)	304.9	513.8	6.43	34.16	146.0	216.8
chair (418K)	191.8	600.7	3.79	26.32	170.0	74.1
room (1,110K)	59.8	82.7	–	3.50	387.0	131.9
counter (1,128K)	36.7	70.1	–	3.03	392.0	131.9
kitchen (1,385K)	26.1	51.2	–	1.99	473.0	161.6
bonsai (1,397K)	55.4	75.0	–	3.38	477.0	140.5
bicycle (1,511K)	87.7	81.5	–	3.89	513.0	100.4
garden (3,870K)	45.1	55.3	–	2.01	1249.0	340.4

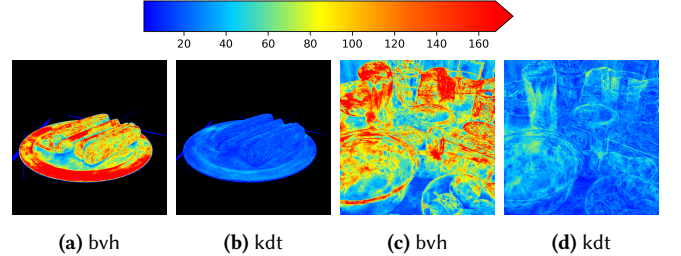


Figure 1: Per-pixel tree traversal costs for the hotdog ((a) & (b)) and counter ((c) & (d)) scenes. The number of ray intersections processed during pixel shading for triangulated (bvh) and analytic (kdt) ellipsoids is color-coded.

scene where performance marginally decreases, Gaussians are distributed relatively sparsely, particularly across background regions, allowing the inherent advantage of rapid hardware-accelerated BVH ray tracing to outweigh its sorting overhead. On the mobile platform, our method achieves a performance speedup ranging from 5.1× to 9.1×, where the performance gains are significantly more pronounced than on the PC platform. Finally, we observe a substantial overall reduction in memory footprint. The memory increase observed in two exceptional scenes is attributed to the highly irregular spatial distribution of their Gaussian splats. Refining the kd-tree construction algorithm to better account for such distributions remains as future work.

References

- J. Barron et al. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 5470–5479.
- M. Jean-Eudes et al. 2026. *Vulkan Gaussian Splatting (Release 2026.2)*. Retrieved June 1, 2026 from https://github.com/nvpro-samples/vk_gaussian_splatting
- B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4 (2023).
- B. Mildenhall et al. 2021. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *Commun. ACM* 65, 1 (2021), 99–106.
- N. Moenne-Loccoz et al. 2024. 3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes. *ACM Trans. Graph.* 43, 6 (2024).
- I. Wald and V. Havran. 2006. On Building Fast Kd-trees for Ray Tracing, and on Doing That in $O(N \log N)$. In *Proc. IEEE Symp. Interact. Ray Tracing*. 61–69.