QUADBOX: ACCELERATING 3D GAUSSIAN SPLATTING WITH GEOMETRY-AWARE BOXES

Xinze Li[1], *Bohan Yang*[1], *Pengxu Chen*[3], *Yiyuan Wang*[1,2]
Hongcheng Luo[4], *Wentao Cheng*[1], *Weifeng Su*[1,5]

[1] Beijing Normal-Hong Kong Baptist University [2] Hong Kong Baptist University
[3] Jilin University [4] Xiaomi
[5] Guangdong Provincial Key Laboratory of Interdisciplinary Research and Application for Data Science
Corresponding author: Wentao Cheng

ABSTRACT

3D Gaussian Splatting (3DGS) has emerged as an advanced technique for real-time novel view synthesis by representing scene geometry and appearance using differentiable Gaussian primitives. However, efficiently computing precise Gaussian–tile intersections remains a critical task in the rasterization pipeline. To this end, we propose QuadBox, a method that leverages four axis-aligned bounding boxes to tightly encapsulate projected Gaussians in a discrete manner. First, we derive a geometry-aware stretching factor that enables the construction of a tile-aligned QuadBox, which covers the elliptical projection and largely excludes irrelevant tiles. Second, we introduce QPass, a single-pass tile traversal algorithm that exhaustively exploits the discrete nature of QuadBox, ensuring that the tile intersection check is performed with simple interval tests. Experiments on public datasets show that our method accelerates the rendering speed of 3DGS by 1.85×.

***Index Terms*—** Gaussian Splatting, Rendering, Bounding box

1. INTRODUCTION

Novel View Synthesis (NVS) aims to generate photorealistic views from novel perspectives, given only a sparse set of captured images. With its ability to combine accurate scene reconstruction and real-time performance, 3D Gaussian Splatting (3DGS) has rapidly become a cornerstone technique in novel view synthesis [1, 2]. On one hand, leveraging the closure properties of elliptical Gaussians under affine transforms and convolution [3], 3DGS accelerates rendering by stacking 2D Gaussians for alpha compositing. On the other hand, the optimized rasterizer leverages the parallelism of modern GPUs to accelerate rendering. Nonetheless, recent efforts aim to further improve 3DGS performance in the face of diverse scene complexities and hardware constraints [4–7].

One straightforward method to accelerate rendering is pruning, which reduces the number of Gaussians in the scene [8–14]. Although pruning reduces computational complexity, it usually undermines visual fidelity, with fewer Gaussians potentially leading to information loss. Another promising paradigm involves identifying redundancies within the rasterizer pipeline to reduce unnecessary computations [15, 16]. In the 3DGS rasterizer, a tile-based design is adopted to enable high parallelism for processing image patches. The initial `preprocess` kernel calculates which tiles intersect with a 2D elliptical Gaussian, and the final `render` kernel uses alpha compositing to assign color to each pixel within intersecting tiles. A key limitation of the original rasterizer lies in its reliance on relaxed bounding boxes, which often include numerous tiles that receive no actual splats during the `render` kernel phase.

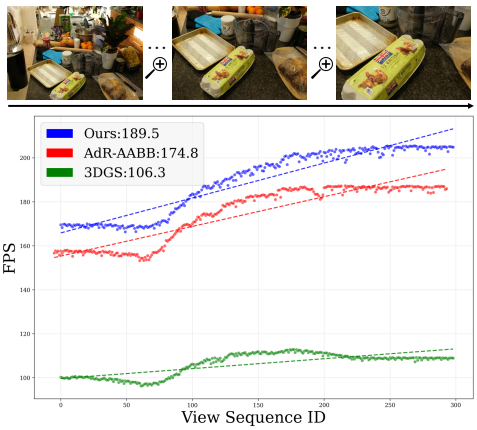


Fig. 1: Rendering speed comparison during zoom-in on an indoor scene. Our method consistently outperforms the Gaussian-tile intersection algorithm (AABB) in AdR-Gaussian [16] and 3DGS in FPS.

The tile redundancy can be alleviated by employing tighter bounding boxes that account for opacity, as demonstrated in AdR-Gaussian [16]. However, its single bounding box design fails to fully capture the anisotropic shape and orientation of the ellipse, which often leads to residual redun-

dancy along the minor axis where tiles may still fall outside the true support region. To address this problem efficiently, we introduce QuadBox, a Gaussian–tile intersection method that strives to fit the elliptical Gaussian using fundamental shapes. The method begins by approximating the elliptical footprint with four axis-aligned bounding boxes, symmetrically positioned around the Gaussian center. These sub-boxes are adaptively stretched to reflect the ellipse's eccentricity and orientation, ensuring tight but conservative coverage. The QuadBox construction is executed once per Gaussian and remains decoupled from tile coverage. Tile-level tests rely solely on integer comparisons, enabling minimal per-tile overhead and strong scalability with scene complexity.

A naive approach that iterates over each sub-box independently introduces redundant tile checks and warp divergence. To address this, we propose QPass, a branch-free single-pass tile traversal algorithm tailored for QuadBox. QPass computes the global tile range with simple integer divisions, and for each scanline, aggregates the active tile interval using fast min/max tests over intersecting sub-boxes. This avoids expensive per-pixel or analytic checks, yielding a deterministic, cache-friendly scan that visits each tile exactly once. Extensive evaluation on the Mip-NeRF 360 [17], Tanks & Temples [18], and Deep Blending [19] datasets demonstrates that our method achieves an average over $1.8\times$ speedup compared to 3DGS [2].

2. PRELIMINARY

3D Gaussian Splatting (3DGS) [2] represents scenes via a set of Gaussian primitives $\{\mathcal{G}_i\}_{i=1}^{N}$. Each primitive is parameterized by a center μ_i, a 3D covariance Σ_i, opacity o_i, and spherical harmonic coefficients for color c_i. The geometry is defined as:

$$\mathcal{G}_i(x) = \exp\left(-\frac{1}{2}(x-\mu_i)^\top \Sigma_i^{-1}(x-\mu_i)\right). \quad (1)$$

To render novel views, 3D Gaussians are projected into 2D space via affine transformation [3]. The splatted opacity at pixel u is given by $\alpha_i = o_i \mathcal{G}_i^{2D}(u)$. Pixel color $C(u)$ is then computed by accumulating contributions in front-to-back order:

$$C(u) = \sum_{i=1}^{N} T_i(u)\,\alpha_i\,c_i, \quad T_i(u) = \prod_{j=1}^{i-1}(1-\alpha_j). \quad (2)$$

The tile-based rasterizer partitions the image plane into 16×16 tiles and executes four key stages. 1) `Preprocess` operates in a Gaussian-parallel manner, projecting primitives and assigning screen-space bounding boxes with side length $6\sqrt{\lambda_{\text{max}}}$, where λ_{max} is the maximum eigenvalue of the 2D covariance. 2) `DuplicateWithKeys` maps Gaussians to their overlapping tiles. 3) `SortPairs` orders these tile-Gaussian pairs by depth. 4) `Render` performs per-pixel alpha blending.

Notably, the bounding boxes used in `preprocess` are conservative, often covering tiles where the Gaussian contribution is negligible. These false positives propagate through the pipeline, incurring unnecessary memory and sorting overheads. To address this, we propose QuadBox, a method that rejects non-contributing tiles during the `preprocess` stage to enhance rendering throughput.

3. METHOD

We introduce QuadBox, a geometric culling method designed to address tile over-approximation. By enclosing each Gaussian with four adaptive axis-aligned bounding boxes (AABBs), QuadBox ensures tight coverage of arbitrary ellipses. This structure serves as the geometric foundation for QPass, our efficient interval-based tile traversal algorithm.

3.1. QuadBox Construction

The construction proceeds in three phases: initialization via opacity filtering, major-axis alignment (DualBox), and adaptive stretching (QuadBox), as shown in Figure 2.

3.1.1. Initialization

To ensure numerical stability during rendering, 3DGS typically ignores Gaussians whose splatted opacity falls below a predefined threshold α_{min}. Leveraging this observation, we follow [16] to filter Gaussians with opacity $\alpha < \alpha_{\text{min}}$ $(1/255)$. This constraint establishes the confidence inequality:

$$2\ln(\frac{o_i}{\alpha_{\text{min}}}) \le (u-\mu_i^{2D})(\Sigma_i^{2D})^{-1}(u-\mu_i^{2D})^\top. \quad (3)$$

Letting $\Lambda = (\Sigma^{2D})^{-1} = [a, b; b, c]$ and centering coordinates as $(x_d, y_d) = u - \mu_i^{2D}$, we reformulate this as the ellipse equation used in subsequent derivation:

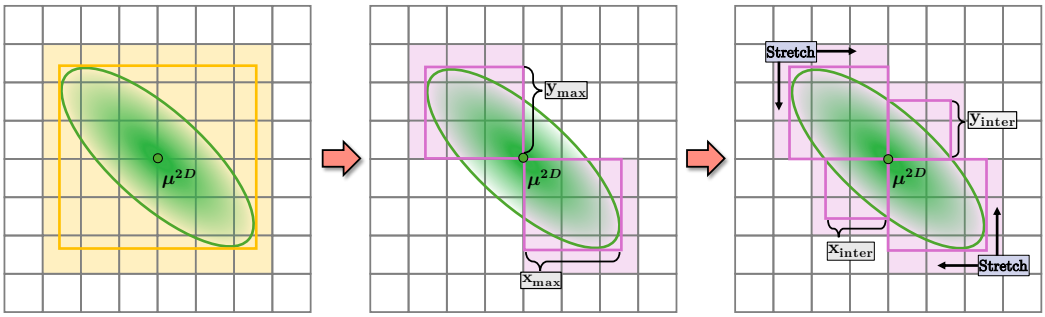
$$F(x_d, y_d) = ax_d^2 + 2bx_d y_d + cy_d^2 - \gamma \le 0, \quad (4)$$

where γ represents the threshold derived from Eq. 3. This region is initially bounded by the tight AdR-AABB [16].

3.1.2. DualBox Extraction

Standard AABBs introduce significant redundancy for elongated ellipses, particularly under diagonal orientations (Fig. 2(a)). To mitigate this, we extract a *DualBox* by retaining only the two quadrants of the AdR-AABB aligned with the ellipse's major axis. This strategy isolates the primary geometric structure, effectively suppressing empty tile coverage along the minor axis.

In the third phase, we aim to tightly enclose the parts of the ellipse not yet covered by the DualBox. To simplify the analysis, we define a local coordinate system by shifting the origin to the center of a ellipse μ^{2D}. As shown in Figure 2(c).



(a) AdR-AABB (b) DualBox (c) QuadBox

Fig. 2: Evolution of QuadBox. (a) AdR-AABB [16]. (b) DualBox splits the region along the major axis. (c) QuadBox adds adaptively stretched sub-boxes to cover the minor axis, derived via closed-form geometric analysis.

Our essential task can be rephrased as determining the intersection points between the ellipse and y-axis in the centered coordinate. The stretching factor we need is the ratio between the distance from the intersection points to μ_y, which can be denoted as y_{inter} and y_{max} of the ellipse. To compute y_{inter}, we simply set $x_d = 0$ in the ellipse definition. As a result, we can construct the stretching factor as:

$$f = \frac{y_{\text{inter}}}{y_{\max}} = \frac{x_{\text{inter}}}{x_{\max}} \tag{5}$$

Setting $x_d = 0$ and $y_d = 0$ in Equation 4 easily gives:

$$c\, y_d^2 = \gamma \Rightarrow y_{\text{inter}} = \sqrt{\gamma/c}, \qquad a\, x_d^2 = \gamma \Rightarrow x_{\text{inter}} = \sqrt{\gamma/a} \tag{6}$$

Critically, we emphasize the importance of crafting not only a straightforward but also high numerical precision formulation that steers clear of excessive use of $\sqrt{\cdot}$ and numerous $\times$ or $\div$ operations. These operands inherently inflate computational overhead and hinder `preprocess` efficiency. In the Appendix, we present a comparative analysis of the computational cost associated with various solution strategies; based on this study, we selected the current closed-form expression that offers the optimal balance between precision and efficiency.

Formally, we can rewrite 4 first by:

$$u^\top (\Sigma^{2D})^{-1} u = \gamma. \tag{7}$$

For simplicity, here we use $\Lambda \in \mathbb{R}^{2\times 2}$ to denote $(\Sigma^{2D})^{-1}$. Also, Λ is notably a positive definite matrix.

For any direction $v \in \mathbb{R}^2$, the maximal support of the ellipse $\max\{v^\top u : \; u^\top C u = \gamma\}$ is

$$\begin{aligned} \max v^\top u &= \sqrt{\gamma\, v^\top \Lambda^{-1} v}\,, \\ u^\star &= \frac{\sqrt{\gamma}}{\sqrt{v^\top \Lambda^{-1} v}}\, \Lambda^{-1} v, \end{aligned} \tag{8}$$

which follows either from the Lagrangian first-order condition or from Cauchy–Schwarz after a Cholesky factorization of Λ. Detailed proof can be found in our appendix.

Choosing $v = e_y = (0,1)^\top$ and $v = e_x = (1,0)^\top$ yields

$$y_{\max} = \sqrt{\gamma\,(\Lambda^{-1})_{22}}\,, \qquad x_{\max} = \sqrt{\gamma\,(\Lambda^{-1})_{11}}\,. \tag{9}$$

For a 2×2 symmetric matrix,

$$\Lambda^{-1} = \frac{1}{ac - b^2} \begin{bmatrix} c & -b \\ -b & a \end{bmatrix} \tag{10}$$

hence

$$y_{\max} = \sqrt{\frac{\gamma\, a}{ac - b^2}}, \qquad x_{\max} = \sqrt{\frac{\gamma\, c}{ac - b^2}}. \tag{11}$$

Combining (11) with the axis intercepts gives the purely algebraic closed form

$$\boxed{f = \sqrt{1 - \frac{b^2}{ac}} = \frac{1}{\sqrt{c\,(\Lambda^{-1})_{22}}} = \frac{1}{\sqrt{a\,(\Lambda^{-1})_{11}}}}. \tag{12}$$

which depends only on the entries of Λ and is independent of the threshold γ and the center translation. Because $\Lambda \succ 0$ implies $a > 0$, $c > 0$, and $ac - b^2 > 0$, the expression satisfies $0 < f \leq 1$, with equality $f = 1$ for circles or when the principal axes align with the coordinate axes.

The reduced tiles from original 3DGS lead to a shortened training and rendering time, also indicating a promising FPS. Moreover, the discrete formulation of QuadBox provides the structural precondition for the following described QPass algorithm, enabling exact tile intersection to be computed via simple integer range checks rather than analytic geometry.

3.2. QPass: QuadBox-based Tile Traversal

3.2.1. QuadBox Stretch

After obtaining a QuadBox for each Gaussian primitive, we need to identify the intersected tiles, thereby establishing the Gaussian-to-tile mapping. This mapping is then inverted

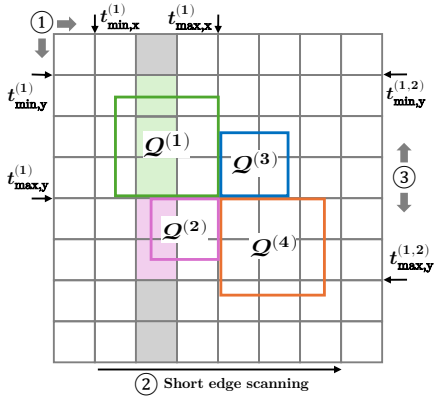


Fig. 3: QPass traversal: (1) Compute global bounds. (2) Scan along the shorter axis. (3) Merge vertical intervals of intersecting sub-boxes to identify active tiles in a single pass.

through the `DuplicateWithKeys` step to enable subsequent tile-level parallel rendering. A naive method to compute the tile coverage of the QuadBox is by looping over the grid-aligned bounds of each sub-box with nested `for` loops, while avoiding redundant visits through conditional checks on overlapping regions. However, independent traversal of each sub-box results in repeated edge tile evaluations, while the branching complexity from variable tile intervals induces significant warp divergence. Moreover, full-grid scanning introduces numerous non-contributing tiles that are later culled by QuadBox, leading to unnecessary computation and degraded rendering performance.

We propose QPass, a single-pass tile traversal algorithm, that avoids redundant tile checks while ensuring complete coverage. At its core, QPass leverages the discrete grid structure and the tight bounds provided by our QuadBox representation. Figure 3 illustrates an exemplary process of how QPass works. Given a QuadBox $\mathcal{Q}$, we partition it into four quadrant-aligned sub-boxes, denoted as $\{\mathcal{Q}^{(i)}\}_{i=1}^{4}$. For each sub-box $\mathcal{Q}^{(i)}$, we first compute its tile-space bounds based on its 2D coordinates, denoted as $\{t^{(i)}_{\min,x}, t^{(i)}_{\max,x}, t^{(i)}_{\min,y}, t^{(i)}_{\max,y}\}_{i=1}^{4}$. These bounds can be directly aggregated to obtain the global tile range of the full QuadBox. To reduce traversal overhead, we select the shorter axis of the global tile range for scanline iteration. Assuming a column-wise traversal, any tile column falling outside the global x-range is immediately skipped. Within each valid column, we identify the sub-boxes whose tile ranges intersect the current column via a fast interval overlap check. The vertical tile range is then determined by aggregating the y-extent of these intersecting sub-boxes using simple min-max operations.

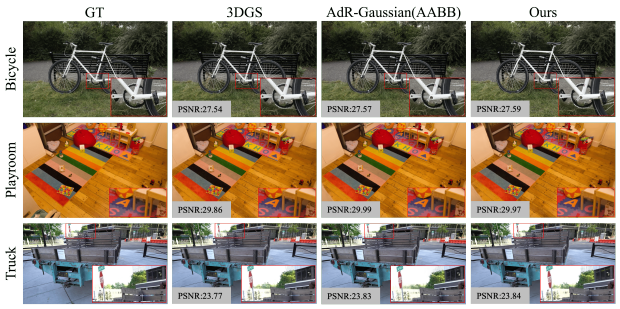


Fig. 4: Qualitative comparison with 3DGS and AdR-AABB. We visualize representative scenes: *bicycle* (Mip-NeRF 360), *playroom* (Deep Blending), and *truck* (Tanks & Temples). Reported PSNR values indicate dataset averages.

4. EXPERIMENTS

4.1. Datasets and Implementation Details

We evaluate on Mip-NeRF 360 [17], Deep Blending [19], and Tanks & Temples [18]. Following standard protocols [2], we use COLMAP poses and sparse point clouds for initialization. Our method is implemented as a custom differentiable rasterizer within the official 3DGS codebase. Experiments are conducted on NVIDIA A100 and RTX 4090 GPUs. Rendering throughput (FPS) is measured via CUDA kernel timing, averaged over three independent trials.

4.2. Comparisons

We benchmark against vanilla 3DGS [2] and AdR-Gaussian [16] (AdR-AABB only). As shown in Table 1 (top), vanilla 3DGS is limited by conservative bounding boxes (180 FPS on Mip-NeRF 360). AdR-AABB improves throughput to 305 FPS via opacity-aware pruning but remains restricted by axis alignment. Our QuadBox method leverages quadrant-aware partitioning to achieve tighter coverage, boosting rendering speed to 322 FPS while maintaining or slightly improving PSNR. Per-scene analysis (Table 2) further confirms that QuadBox consistently outperforms AdR-AABB across diverse indoor and outdoor scenes, validating its robustness in handling complex geometries. Qualitative comparisons are shown in Figure 4.

Based on the results in Table 2, we further perform a per-scene comparison between AdR-AABB and our method to assess robustness across diverse environments. Our method consistently outperforms AdR-AABB in rendering speed across nearly all scenes. This suggests that our QuadBox-based tile pruning achieves better runtime efficiency without compromising image fidelity or requiring more Gaussians. The performance gain holds across both indoor and outdoor environments, highlighting the general robustness and scalability of our rasterization strategy. A qualitative comparison is shown in Figure 4.

Table 1: Comparison of acceleration strategies on NVIDIA A100. Top: Core intersection modules on vanilla 3DGS. Bottom: Plug-and-play integration with recent variants. "3DGS + Ours" denotes our full pipeline. The **bold** results represent the best performance of each sub-comparison.

Method	MipNeRF-360					Deep Blending					Tanks and Temples				
	$N_{GS}\downarrow$	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	$N_{GS}\downarrow$	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	$N_{GS}\downarrow$	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑
Gaussian-tile Intersection															
3DGS	2.76M	27.54	0.817	0.216	180	**2.48M**	29.86	0.908	0.242	173	**1.57M**	23.77	0.853	0.170	214
AdR-AABB	**2.73M**	27.57	0.817	**0.215**	302	2.55M	**29.99**	0.908	0.242	348	**1.57M**	23.83	**0.854**	0.170	319
3DGS + Ours	2.77M	**27.59**	**0.818**	**0.215**	**322**	2.57M	29.97	**0.908**	**0.242**	**379**	1.60M	**23.84**	0.854	**0.169**	**335**
Plug-and-Play Experiments															
DashGaussian	**2.26M**	27.46	0.808	0.233	142	**1.81M**	29.86	0.906	0.253	159	**1.17M**	24.11	**0.852**	0.184	197
+Ours	2.37M	**27.50**	**0.811**	**0.230**	**305**	1.91M	**29.83**	**0.906**	**0.251**	**331**	1.20M	**24.13**	**0.852**	**0.183**	**364**
Compact-3DGS	**1.43M**	27.03	0.800	**0.243**	221	**1.04M**	29.94	0.906	**0.259**	296	**0.84M**	23.33	**0.852**	**0.201**	282
+Ours	1.45M	**27.06**	**0.808**	0.247	**441**	1.07M	**29.95**	**0.908**	0.261	**684**	0.85M	**23.34**	**0.835**	0.206	**463**

Method	AdR-AABB			QuadBox		
	$N_{GS}\downarrow$	PSNR ↑	FPS ↑	$N_{GS}\downarrow$	PSNR ↑	FPS ↑
bicycle	4.89M	25.31	199	5.03M	25.23	**206**
flowers	2.86M	21.61	360	2.95M	21.61	**367**
garden	4.21M	27.51	241	4.18M	27.48	**251**
stump	4.30M	26.64	309	4.30M	26.69	**318**
treehill	3.18M	22.61	312	3.31M	22.58	**318**
room	1.35M	31.69	324	1.35M	31.67	**363**
counter	1.10M	29.09	312	1.11M	29.09	**347**
kitchen	1.62M	31.61	242	1.62M	31.51	**261**
bonsai	1.10M	32.07	423	1.09M	32.43	**464**

Table 2: Per-scene comparison between AdR-AABB and our proposed QuadBox method on Mip-NeRF 360 (measured with NVIDIA A100). The **bold** results show the best FPS performance.

4.3. Plug-and-Play Results

We integrate our rasterizer into two state-of-the-art backbones to demonstrate generality (Table 1, bottom). DashGaussian [20]: Integration yields a $> 1.85\times$ speedup (peaking at 364 FPS) with consistent quality gains, proving compatibility with momentum-based densification. Compact-3DGS [10]: Our method accelerates this memory-efficient model by $> 1.64\times$ (up to 684 FPS) while improving SSIM/PSNR. These results verify that our geometry-aware culling enhances efficiency across different optimization strategies without sacrificing model compactness.

Method	$N_{GS}\downarrow$	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑
3DGS	**2.74M**	27.54	0.815	0.216	247
3DGS+AdR-AABB	2.78M	<u>27.59</u>	<u>0.817</u>	<u>0.215</u>	425
3DGS+DualBox	2.79M	27.43	0.814	0.217	**467**
3DGS+QuadBox†	2.78M	**27.60**	**0.818**	**0.214**	413
3DGS+QuadBox	<u>2.77M</u>	<u>27.59</u>	<u>0.817</u>	**0.214**	<u>460</u>

Table 3: The ablation study of our proposed QuadBox (measured with NVIDIA 4090). The † denotes our proposed method without QPass traversal. The **bold** and <u>underlined</u> results show the best and second best performance.

4.4. Ablation Study

Table 3 evaluates module contributions on an RTX 4090. AdR-AABB [16] accelerates rendering but offers coarse coverage. *DualBox* boosts FPS via aggressive culling but degrades quality due to symmetric approximation errors. The intermediate QuadBox† recovers this quality loss, validating the necessity of auxiliary sub-boxes. Finally, our full QuadBox with QPass achieves the optimal balance, surpassing QuadBox† in speed and outperforming AdR-AABB in both efficiency (460 vs 425 FPS) and fidelity. This confirms QuadBox as a robust, high-performance intersection strategy.

5. CONCLUSION

We present a novel tile-based rasterization strategy that rethinks how Gaussians interact with discrete grids. By introducing QuadBox, a quadrants-aware bounding scheme, and QPass, a branch-free traversal mechanism, we effectively reduce redundant tile checks while preserving full coverage. Unlike existing approaches that rely on uniform heuristics or coarse approximations, our method adapts to the geometry and anisotropy of each Gaussian, enabling precise culling with minimal overhead. Extensive experiments across high-complexity scenes demonstrate consistent acceleration over state-of-the-art baselines, with either improved or retained rendering quality. Our design is orthogonal to most existing Gaussian splatting pipelines and can be readily integrated to enhance scalability and performance under dense visibility conditions.

6. ACKNOWLEDGEMENT

This work was supported by the National Natural Science Foundation of China (Grant No. 62306154).

7. REFERENCES

[1] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng,

"Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[2] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis, "3d gaussian splatting for real-time radiance field rendering.," *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139–1, 2023.

[3] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross, "Ewa splatting," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 223–238, 2002.

[4] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger, "Mip-splatting: Alias-free 3d gaussian splatting," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 19447–19456.

[5] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou, "Absgs: Recovering fine details in 3d gaussian splatting," in *Proceedings of the 32nd ACM International Conference on Multimedia*, 2024, pp. 1053–1061.

[6] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa, "Plenoxels: Radiance fields without neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 5501–5510.

[7] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai, "Scaffold-gs: Structured 3d gaussians for view-adaptive rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 20654–20664.

[8] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al., "Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps," *Advances in neural information processing systems*, vol. 37, pp. 140138–140158, 2024.

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

[10] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park, "Compact 3d gaussian representation for radiance field," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 21719–21728.

[11] Zhaoliang Zhang, Tianchen Song, Yongjae Lee, Li Yang, Cheng Peng, Rama Chellappa, and Deliang Fan, "Lp-3dgs: Learning to prune 3d gaussian splatting," *Advances in Neural Information Processing Systems*, vol. 37, pp. 122434–122457, 2024.

[12] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre, "Taming 3dgs: High-quality radiance fields with limited resources," in *SIGGRAPH Asia 2024 Conference Papers*, 2024, pp. 1–11.

[13] Zheng Zhang, Wenbo Hu, Yixing Lao, Tong He, and Hengshuang Zhao, "Pixel-gs: Density control with pixel-aware gradient for 3d gaussian splatting," in *European Conference on Computer Vision*. Springer, 2024, pp. 326–342.

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

[15] Guofeng Feng, Siyan Chen, Rong Fu, Zimu Liao, Yi Wang, Tao Liu, Boni Hu, Linning Xu, Zhilin Pei, Hengjie Li, et al., "Flashgs: Efficient 3d gaussian splatting for large-scale and high-resolution rendering," in *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025, pp. 26652–26662.

[16] Xinzhe Wang, Ran Yi, and Lizhuang Ma, "Adr-gaussian: Accelerating gaussian splatting with adaptive radius," in *SIGGRAPH Asia 2024 Conference Papers*, 2024, pp. 1–10.

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

[18] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–13, 2017.

[19] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (ToG)*, vol. 37, no. 6, pp. 1–15, 2018.

[20] Youyu Chen, Junjun Jiang, Kui Jiang, Xiao Tang, Zhihao Li, Xianming Liu, and Yinyu Nie, "Dashgaussian: Optimizing 3d gaussian splatting in 200 seconds," in *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025, pp. 11146–11155.