AdR-Gaussian: Accelerating Gaussian Splatting with Adaptive Radius

XINZHE WANG*, Shanghai Jiao Tong University, China
RAN YI*, Shanghai Jiao Tong University, China
LIZHUANG MA†, Dept. of Computer Science and Engineering, Shanghai Jiao Tong University, China

Fig. 1. Our method achieves a significant improvement in rendering speed for the 3D Gaussian Splatting model with equivalent or better quality. The key to this performance is the lossless early culling of Gaussian-Tile pairs based on adaptive radius of projected Gaussians, which narrows the rendering range for each Gaussian by removing tiles with low splatting opacity. Additionally, we propose a load balancing approach to further accelerate rendering.

3D Gaussian Splatting (3DGS) is a recent explicit 3D representation that has achieved high-quality reconstruction and real-time rendering of complex scenes. However, the rasterization pipeline still suffers from unnecessary overhead resulting from avoidable serial Gaussian culling, and uneven load due to the distinct number of Gaussian to be rendered across pixels, which hinders wider promotion and application of 3DGS. In order to accelerate Gaussian splatting, we propose *AdR-Gaussian*, which moves part of serial culling in *Render* stage into the earlier *Preprocess* stage to enable parallel culling, employing adaptive radius to narrow the rendering pixel range for each Gaussian, and introduces a load balancing method to minimize thread waiting time during the pixel-parallel rendering. Our contributions are threefold, achieving a rendering speed of 310% while maintaining equivalent or even better quality than the state-of-the-art. Firstly, we propose to early cull Gaussian-Tile pairs of low splatting opacity based on an adaptive radius in the Gaussian-parallel *Preprocess* stage, which reduces the number of affected tile through the Gaussian bounding circle, thus reducing unnecessary overhead and achieving faster rendering speed. Secondly, we further propose early culling based on axis-aligned bounding box for Gaussian splatting, which achieves a more significant reduction in ineffective expenses by accurately calculating the Gaussian size in the 2D directions. Thirdly, we propose a balancing algorithm for pixel thread load, which compresses the information of heavy-load pixels to reduce thread waiting time, and enhance information of light-load pixels to hedge against rendering quality loss. Experiments on three datasets demonstrate that our algorithm can significantly improve the Gaussian Splatting rendering speed.

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

Additional Key Words and Phrases: Novel view synthesis, Guassian splatting, Real-time rendering, Bounding box

ACM Reference Format:
Xinzhe Wang, Ran Yi, and Lizhuang Ma. 2024. AdR-Gaussian: Accelerating Gaussian Splatting with Adaptive Radius. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24), December 03–06, 2024, Tokyo, Japan.* ACM, New York, NY, USA, Article 118, 10 pages. https://doi.org/10.1145/3680528.3687675

1 Introduction

Novel view synthesis (NVS) aims to generate photorealistic rendering results of novel views given a set of input views, which has attracted significant attention due to its wide applications in various domains, including model design [Chen et al. 2023a; Tang et al. 2023], autonomous driving [Cao et al. 2024; Matsuki et al. 2023], and virtual reality [Qian et al. 2023; Wang et al. 2023]. The 3D Gaussian Splatting (3DGS) model [Kerbl et al. 2023] is a recent 3D representation that employs a set of 3D Gaussian ellipsoids to model 3D scenes, achieving high-quality real-time rendering of complex scenes. However, the Gaussian rasterization pipeline suffers from unnecessary overhead resulting from avoidable serial Gaussian culling and uneven load due to the distinct number of Gaussian to be rendered across pixels, which limits the rendering speed of 3D Gaussian and hinders its wider applications.

The Gaussian rasterization pipeline currently suffers from unnecessary overhead and uneven load due to the following reasons, which are mainly related to the *Render* stage that performs pixel-parallel point-based rendering in the pipeline: (1) *Unnecessary overhead of serial culling*: Gaussian-Pixel pairs with low splatting opacity

*Both authors contributed equally to this research.
†Corresponding author.

Authors' Contact Information: Xinzhe Wang, Shanghai Jiao Tong University, China, hiroxzwang@sjtu.edu.cn; Ran Yi, Shanghai Jiao Tong University, China, ranyi@sjtu.edu.cn; Lizhuang Ma, Dept. of Computer Science and Engineering, Shanghai Jiao Tong University, China, ma-lz@cs.sjtu.edu.cn.

SA Conference Papers '24, December 03–06, 2024, Tokyo, Japan

ACM ISBN 979-8-4007-1131-2/24/12
https://doi.org/10.1145/3680528.3687675

are culled serially during the *Render* stage, but some of the serial culling could be performed in parallel, thus current culling method results in a significant performance penalty for the Gaussian-serial-based alpha blending. (2) *Uneven load across pixel threads*: Since the *Render* stage calculates the color of each pixel in parallel, and the number of Gaussian to be rendered serially for each pixel are different, different pixel threads face distinctly different loads (computational time), which causes long thread waiting time and low rendering efficiency. The existing approaches [Fan et al. 2023; Girish et al. 2023; Hamdi et al. 2024; Lee et al. 2023] that accelerate rendering by reducing the total number of Gaussian sacrifices the rendering quality, and fails to effectively handle the aforementioned problems, leaving room for further improvements of rendering speed.

To address these issues, we propose *AdR-Gaussian*, which employs lossless early culling with adaptive radius and axis-aligned bounding box tailored for Gaussian Splatting to narrow the rendering pixel range of each Gaussian, and proposes a load balancing method to minimize thread waiting time. Our *AdR-Gaussian* consists of two phases: a early-culling phase and a load balancing phase. Firstly, we propose to move part of serial culling in *Render* stage earlier into parallel culling in *Preprocess* stage. We early cull Gaussian-Tile pairs with low splatting opacity during *Preprocess* stage, which is conducted in parallel for each Gaussian. Specifically, we propose *adaptive radius* and *axis-aligned bounding box* for Gaussian Splatting to early cull Gaussian-Tile pairs, which achieves acceleration of Gaussian rendering without rendering quality loss. Furthermore, to address the issue of uneven computational costs across different pixel threads in Gaussian rendering, we propose a *load balancing* algorithm for pixel threads. By quantifying the variance of the number of rendered Gaussian required for each pixel, we reduce the number of Gaussian for heavy-load pixels to reduce thread waiting time, and increase that for light-load pixels to hedge against the quality loss, achieving higher rendering efficiency.

In summary, we make the following contributions:

- We propose *AdR-Gaussian*, a novel Gaussian Splatting rendering acceleration method, which moves part of serial culling in *Render* stage earlier into parallel culling in *Preprocess* stage, and balances load across different pixel threads.
- We early cull Gaussian-Tile pairs with low splatting opacity during the Gaussian-parallel *Preprocess* stage based on adaptive radius, defined as the bounding circle radius of an ellipse constructed by minimum splatting opacity, which achieves acceleration without rendering quality loss.
- We propose early culling based on axis-aligned bounding box tailored for Gaussian Splatting, which achieves different extents of culling in horizontal and vertical directions, and further improves culling efficiency.
- We propose a load balancing algorithm for pixel thread, which reduces the variance of the number of Gaussian to be rendered serially for each pixel, to reduce thread waiting time and further accelerate rendering.

Experiments on three datasets demonstrate our approach significantly enhances the Gaussian Splatting rendering speed, achieving 310% acceleration on average, while achieving equivalent or superior rendering quality. Specifically, for complex scenes in the Mip-NeRF360 dataset [Barron et al. 2022], our method attains an average 590FPS rendering speed on a single NVIDIA RTX3090 GPU.

2 Related Works

We first present a brief overview of traditional novel view synthesis approaches, followed by a discussion on the neural radiance fields (NeRF) and the 3D Gaussian Splatting (3DGS) model. Given that both of the latter two fields are extensive, we focus on the rendering acceleration methods, which are most pertinent to our work. For a more comprehensive information and analysis, readers are referred to recent surveys [Chen and Wang 2024; Tewari et al. 2022].

2.1 Traditional Novel View Synthesis

Early novel view synthesis methods often rely on physical principles, such as light field sampling interpolation [Gortler et al. 1996; Levoy and Hanrahan 1996], mesh representation [Buehler et al. 2001; Debevec et al. 1996], and voxel representation [Flynn et al. 2019; Mildenhall et al. 2019]. However, these methods are based on discrete sampling and suffer from issues including poor reconstruction quality, dependence on initialization data, and high spatial complexity. With the development of neural networks, methods such as signed distance fields [Jiang et al. 2020; Park et al. 2019], 3D occupancy fields [Niemeyer et al. 2020], and differentiable rendering [Sitzmann et al. 2019] have been introduced to model 3D scenes from 2D images through implicit representation. Nevertheless, for the synthesis of high-resolution and high-quality novel views in complex geometric scenes, all these methods proved challenging until the introduction of NeRF.

2.2 Fast NeRF Rendering

NeRF [Mildenhall et al. 2021] employs 5D parameters including position and camera direction to obtain RGBA information via multi-layer perceptron (MLP). However, due to the use of volume rendering approach and the complexity of MLP, NeRF is inefficient, thus lots of research focus on accelerating its rendering speed.

NSVF [Liu et al. 2020] introduces Octree with empty space skipping and early ray termination strategy to minimize invalid sampling. RT-Octree [Shu et al. 2023] uses low-SPP Monte Carlo rendering to reduce arbitrary sampling followed by denoising. KiloNeRF [Reiser et al. 2021] distills the large MLP into thousands of smaller MLPs, thereby reducing MLP query costs. Other approaches cache information using data structures [Hedman et al. 2021; Yu et al. 2021] or hash encoding [Hu et al. 2023; Reiser et al. 2023], further reducing query costs by minimizing MLP usage. Furthermore, some work also employs GPU-friendly implementations [Chen et al. 2023b] to achieve even faster rendering speeds.

However, these real-time NeRF rendering often sacrifice other metrics, such as slower training caused by the establishing and updating of data structures. In contrast, the recently proposed 3DGS model achieves high-quality and real-time rendering, and fast training through differentiable, GPU-friendly Gaussian ellipsoids, and our method aims to further improve the rendering speed of 3DGS.

2.3 Fast 3DGS Rendering

The 3DGS model [Kerbl et al. 2023] is a recent explicit 3D representation employing a set of 3D Gaussian ellipsoids, which simultaneously achieve high-quality rendering, competitive training time, and real-time rendering. The 3DGS model trains and fits the scene via Gaussian's differentiability, and achieves efficient rendering through Gaussian's explicit structure. Owing to its high-quality real-time rendering capability, a large amount of application researches have emerged [Szymanowicz et al. 2023; Wu et al. 2023; Zhou et al. 2024], while a few efforts focus on optimizing its storage [Niedermayr et al. 2024] or synthesis quality [Yu et al. 2023].

Some recent researches accelerate the rendering speed of 3DGS by reducing the number of Gaussian. EAGLES [Girish et al. 2023] suppresses the frequency of Gaussian densification via convergence curve, thereby directly decelerating the growth rate of Gaussian number. GES [Hamdi et al. 2024] introduces learning tendency from low frequency to high frequency, limiting the creation of Gaussian for high-frequency information. Additionally, other methods reduce the overall number of Gaussian by pruning those with lower significance. LightGaussian [Fan et al. 2023] assesses Gaussian importance based on its volume, opacity, and the number of pixel affected, under which high-ratio pruning retains decent quality. C3DGS [Lee et al. 2023] employs Gaussian scaling and opacity to generate removal mask, and incorporates masking loss into the loss function to strike a balance between synthesis quality and rendering speed.

However, current fast 3DGS rendering methods lack attention to the optimization of rasterization pipeline of 3DGS, which often result in limited improvements in rendering speed and degenerated quality caused by the reduction of Gaussian number. In contrast, our *AdR-Gaussian* focuses on optimizing the rasterization pipeline of 3DGS, and can significantly accelerate rendering with equivalent or better quality.

3 Preliminaries and Analysis

3.1 3D Gaussian Representation

As an explicit 3D representation, 3DGS utilizes a set of 3D Gaussian ellipsoids with geometric and material properties to model the scene.

The geometry of each Gaussian is defined by its center position $\mathbf{X}_0$, and a 3D covariance matrix Σ defined in world space [Zwicker et al. 2001]:

$$G(\mathbf{X}) = e^{-\frac{1}{2}(\mathbf{X}-\mathbf{X}_0)^T\Sigma^{-1}(\mathbf{X}-\mathbf{X}_0)}, \tag{1}$$

where $\mathbf{X}$ is a sampling point and the covariance matrix is further decomposed into a scaling and a rotation matrix.

The material information of each Gaussian includes opacity attribute σ, and spherical harmonics (SH) (representing color). To render from a specific perspective, based on the concept of Elliptic Weighted Average (EWA) [Zwicker et al. 2002], the splatting opacity α for each Gaussian-Pixel pair is calculated by the opacity σ, Gaussian-pixel 2D distance $\mathbf{x}$ and projected covariance matrix Σ':

$$\alpha = \sigma e^{-\frac{1}{2}\mathbf{x}^T\Sigma'^{-1}\mathbf{x}}. \tag{2}$$

With Gaussian color c calculated from SH, and splatting opacity α, the color of each pixel is calculated by alpha blending N related Gaussians:

$$C = \sum_{i\in N} c_i\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j). \tag{3}$$

Such an alpha blending idea is based on the neural point-based approach [Kopanas et al. 2021], and has achieved very high efficiency through a tile-based soft rasterizer.

3.2 Gaussian Rasterization

Inspired by the soft rasterization approaches [Lassner and Zollhöfer 2021], the Gaussian rasterization adopts a tile-based and sorting renderer. The screen is firstly splitted into 16×16 tiles. And then each Gaussian is linked to the tiles it overlaps with. The Gaussians in each tile are sorted by view space depth, and then the color is calculated by alpha blending.

3.2.1 Rasterization Pipeline. The rasterization pipeline consists of six stages: *Preprocess, InclusiveSum, DuplicateWithKeys, SortPairs, IdentifyTileRanges,* and *Render*. Their roles are as follows:

1) *Preprocess* stage: preprocesses each Gaussian in parallel to obtain the perspective-related data needed for rendering.

2-3) *InclusiveSum and DuplicateWithKeys* stages: process each Gaussian in parallel. For each Gaussian, these stages obtain the information of which tiles it covers.

4-5) *SortPairs and IdentifyTileRanges* stages: aim to sort the Gaussians in each tile by depth, and determine the start and end indices of the Gaussian within each tile for rendering.

6) *Render* stage: calculates the color of each pixel in parallel, using a point-based approach to sequentially render the Gaussians within the corresponding tile by alpha blending.

3.2.2 Rasterization Costs Analysis. The computational cost $\mathcal{E}$ required in the Gaussian rasterization pipeline can be categorized into three parts, depending on the degree of GPU parallelism:

$$\mathcal{E} = \mathcal{E}_g + \mathcal{E}_n + \mathcal{E}_p, \tag{4}$$

where (1) $\mathcal{E}_g$ denotes the Gaussian-level parallel cost of the 1-3th stages, for preprocessing Gaussians to obtain their color and other perspective related attributes; (2) $\mathcal{E}_n$ represents the Gaussian-Tile pair parallel cost of the 4-5th stages, for identifying Gaussian range for each tile; and (3) $\mathcal{E}_p$ represents the pixel-level parallel cost of the 6-th stage, for the point-based rendering.

Due to the serial alpha blending of Gaussians in the *Render* stage, and the complexity of calculating the splatting color for each Gaussian-Pixel pair, this stage has the largest computational time proportion, *i.e.,* $\mathcal{E}_p$ takes the largest proportion (over 50%) in $\mathcal{E}$. To accelerate the *Render* stage, we propose *AdR-Gaussian*, which culls pairs of Gaussian-Tile with low splatting opacity earlier in *Preprocess* stage, and balances the computational costs of pixel thread.

4 Method

4.1 Overview

Our main goal is to reduce the computational cost $\mathcal{E}$ of the Gaussian rasterization pipeline (Sec. 3.2), which can be decomposed into three types of costs as in Eq. (4). Among these three types of costs, the pixel-level parallel cost $\mathcal{E}_p$ of the *Render* stage (Sec. 3.2.1) takes the largest proportion. This is because the pixel-parallel *Render* stage

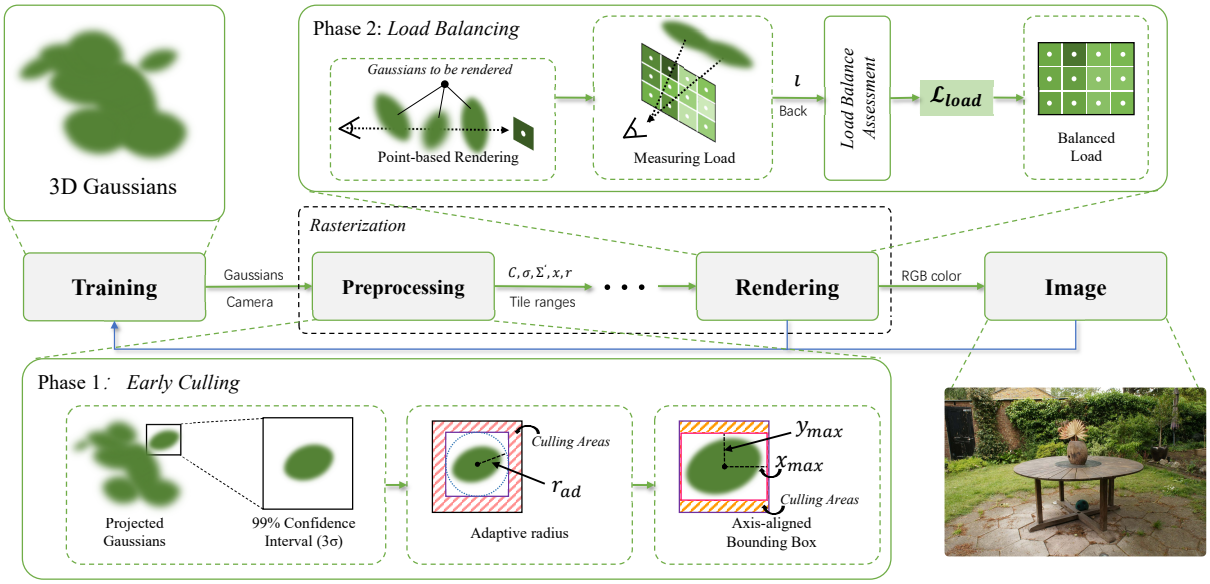

Fig. 2. Pipeline overview. We propose *AdR-Gaussian*, designed to accelerate the rendering process of 3DGS. In this approach, we propose to move part of serial culling in *Render* stage earlier into parallel culling in *Preprocess* stage. We early cull Gaussian-Tile with low splatting opacity based on adaptive radius, which is the bounding circle radius of the ellipse constructed by minimum splatting opacity. Furthermore, we employ *axis-aligned bounding box* tailored for Gaussian Splatting to improve culling efficiency. To address the uneven load issue, we propose a load balancing algorithm for pixel threads, which reduces the number of Gaussian for heavy-load pixels to reduce thread waiting time, and increases that for light-load pixels to hedge against quality loss.

has a high degree of seriality in the handling of Gaussians, *i.e.,* to calculate pixel color based on alpha blending, it necessitates the sequential traversal of all Gaussians within the corresponding tile. Therefore, we aim to accelerate the pixel-parallel *Render* stage, by moving part of serial culling earlier into Gaussian-parallel stages, and balancing the computational costs across parallel pixel threads.

As shown in Fig. 2, the pipeline of our *AdR-Gaussian* consists of two distinct phases:

Phase 1: Early Culling (Sec. 4.2, 4.3). We move part of serial culling in *Render* stage earlier into parallel culling in *Preprocess* stage. We early cull Gaussian-Tile pairs with low splatting opacity during *Preprocess* stage, which is conducted in parallel for each Gaussian, and reduces the number of Gaussian-Tile to be processed in the later stages. Specifically, we propose two early culling algorithms: 1) we early cull pairs of Gaussian-Tile with low splatting opacity based on adaptive radius, which is the radius of the bounding circle of an ellipse constructed by minimum splatting opacity. 2) Furthermore, we employ axis-aligned bounding box to improve culling efficiency and achieve higher rendering speed.

Phase 2: Load balancing (Sec. 4.4). For *Render* stage, since the color of each pixel is calculated in parallel, and the computational costs of different pixel threads are uneven, the overall efficiency largely depends on heavy-load pixel threads. We propose to balance load across pixel threads, reducing the number of rendered Gaussian for heavy-load pixels to minimize thread waiting, and increasing that for light-load pixels to hedge against quality loss, which achieves more efficient rendering with equivalent or better quality.

4.2 Early Culling with Adaptive Radius

Since the Gaussians in each tile are rendered in serial, the *Render* stage takes the largest load proportion, which motivates us to accelerate this stage. We observe that the cull of Gaussians (removing Gaussians of low splatting opacity) are conducted in *Render* stage: The splatting opacity of each Gaussian is calculated by Eq. (2), based on Gaussian-Pixel 2D distance $\mathbf{x}$, which is pixel-related, and pixel information can only be obtained from *Render* stage. However, since the process of each Gaussian in *Render* is serial, which makes the culling also serial. We regard the culling process can be parallel, and propose to move part of the culling earlier, into previous *Preprocess* stage to avoid serial culling and enable parallel culling.

Our goal is to early cull pairs of Gaussian-Tile with extremely low splatting opacity in parallel in *Preprocess* stage, to avoid unnecessary serial processing and achieve faster rendering speed. To achieve this, we propose adaptive radius for projected Gaussians, which computes the bounding circle for each Gaussian, and early culls tiles out of bounding circle during the *Preprocess* stage, where Gaussians are processed in parallel. Compared to the original rendering range based on 99% confidence interval (black square), the rendering range of the bounding circle is shown as the purple square in Fig. 3a. Owing to the constraints of the rasterization pipeline, the actual rendering extent is the circumscribed square of the bounding circle.

To get the bounding circle, we compute projected Gaussian's adaptive radius based on the splatting opacity, instead of the original 99% confidence interval (calculated from the standard deviation of a 2D Gaussian distribution). Although the splatting opacity α is yet unknown during the *Preprocess* stage, we know a lower bound of

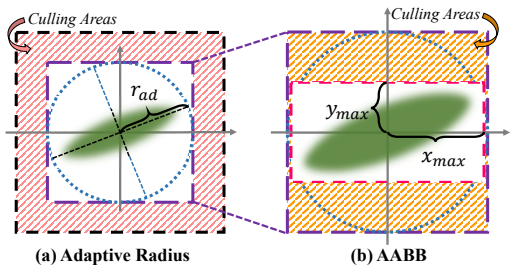

Fig. 3. The rendering range of early culling approaches: (a) Adaptive Radius, (b) AABB for Gaussian. The original rendering range, *i.e.,* the bounding box of 99% confidence interval, is shown as the black square, which will be further culled in *Render* stage based on splatting opacity. We propose to early cull in *Preprocess* stage with (1) adaptive radius, which is the bounding circle (blue) radius of an ellipse constructed by minimum splatting opacity, and remove the redundant red areas, resulting in a smaller rendering range (in purple); and (2) we further propose to cull with axis-aligned bounding box (in rose red) for Gaussian Splatting, which achieves different extents of culling in 2D directions, significantly accelerating rendering speed.

splatting opacity, which is a predefined minimum splatting opacity constant α_{low} set in the original 3DGS model: $\alpha_i \geq \alpha_{low}$, where α_i is the splatting opacity of Gaussian i at the current pixel. By substituting Eq. (2) into this inequality, we get the following expression:

$$\mathbf{x}^T {\Sigma'}^{-1} \mathbf{x} \leq 2ln\left(\frac{\sigma_i}{\alpha_{low}}\right). \tag{5}$$

Since 2D distance $\mathbf{x}$ can be expressed as its 1D components (x, y), and covariance Σ' can be expressed as $[\Sigma'_X, \Sigma'_{XY}; \Sigma'_{XY}, \Sigma'_Y]$, inequality Eq. (5) represents an ellipse, wherein x, y serve as variables:

$$Ax^2 + By^2 + Cxy + D \leq 0, \tag{6}$$

where

$$A = \Sigma'_Y,\ B = \Sigma'_X,\ C = -2\Sigma'_{XY},$$
$$D = -2\left(\Sigma'_X\Sigma'_Y - {\Sigma'_{XY}}^2\right) ln\left(\frac{\sigma_i}{\alpha_{low}}\right).$$

We then define the adaptive radius of a projected Gaussian as the bounding circle radius of this ellipse, *i.e.,* half of the ellipse's major axis length. According to the standard equation of ellipse, the half length of an ellipse's major axis is equal to the square root of the reciprocal of the smaller eigenvalue λ_{ad} of the ellipse's covariance matrix Σ_{ad}: $r_{ad} = \sqrt{1/\lambda_{ad}} = \sqrt{2D/(-(A+B) + \sqrt{(A-B)^2 + C^2})}$. By substituting coefficients, adaptive radius is formulated as follows:

$$r_{ad} = \sqrt{2\lambda'_{max} ln\left(\frac{\sigma_i}{\alpha_{low}}\right)}, \tag{7}$$

where λ'_{max} is the larger eigenvalue of the projected covariance Σ', which has been computed earlier for the 99% confidence interval (Detailed formula are presented in Supplementary materials).

Finally, there may be situations where the Gaussian opacity is high or the predefined minimum splatting opacity is low, resulting in an adaptive radius r_{ad} that exceeds the original radius r_o determined based on the 99% confidence interval. Therefore, the final value of the 2D Gaussian radius should be the minimum of these two values:

$$r = min(r_{ad}, r_o). \tag{8}$$

By performing early culling based on adaptive radius in *Preprocess* stage in a parallel manner, we can save some computation costs caused by serial culling during *Render* stage.

4.3 Early Culling with Axis-aligned Bounding Box for Gaussian Splatting

Although the adaptive radius significantly reduces unnecessary overhead in Gaussian splatting rasterization, it cannot effectively cull pairs of Gaussian-Tile with low splatting opacity in the minor axis direction (Fig. 3b, regions marked in orange), as the radius is determined by the major axis. To address this issue, we propose axis-aligned bounding box for Gaussian splatting, which achieves a more significant reduction in ineffective expenses by accurately calculating the Gaussian size in the 2D directions.

For a projected Gaussian, we further cull the tiles based on the axis-aligned bounding box of the ellipse (Eq. (6)) constructed by the minimum splatting opacity α_{low} . Specifically, half of the bounding box's width w and height h equal to the maximum values in the two coordinate directions of the ellipse, x_{max} and y_{max}, respectively. We first define a function F for the ellipse: $F = Ax^2 + By^2 + Cxy + D$.

Based on the ellipse function, we calculate the corresponding partial derivatives in two coordinate directions respectively:

$$\frac{\partial F}{\partial x} = 2Ax + Cy,\ \frac{\partial F}{\partial y} = 2By + Cx. \tag{9}$$

Based on geometric properties of ellipse, when the ellipse function's partial derivative with respect to one coordinate direction is 0, the coordinate value in another direction attains its extremum. To get the extremum of the ellipse on both coordinates, we let the two derivatives to be 0, and substitute the two coordinate relationships into the ellipse function, and solve x_{max} and y_{max} as follows:

$$x_{max} = \sqrt{2\Sigma'_X ln\left(\frac{\sigma_i}{\alpha_{low}}\right)},\ y_{max} = \sqrt{2\Sigma'_Y ln\left(\frac{\sigma_i}{\alpha_{low}}\right)}. \tag{10}$$

Compared to bounding circle with adaptive radius, the axis-aligned bounding box for Gaussian splatting can achieve different extents of culling in horizontal and vertical directions, thereby obtaining different tile ranges for the two directions. Furthermore, similar to the adaptive radius for bounding circle, we take the original radius as the upper limit:

$$r_x = min\left(x_{max}, r_o\right),\ r_y = min\left(y_{max}, r_o\right). \tag{11}$$

With this approach, the Gaussian rendering range can align with the axis-aligned bounding box, significantly reducing rendering overhead and achieving more efficient rendering.

4.4 Load Balancing for Pixel-parallel Splatting

Uneven Load Issue. Based on the point-based rendering, the *Render* stage of Gaussian rasterization has a pixel-parallel and Gaussian-serial architecture: the color calculation of each pixel is conducted in parallel, each pixel corresponding to a pixel thread; while in each pixel thread, the Gaussians are rendered sequentially. Although the

Fig. 4. Visualization of pixel load in Truck dataset. Brighter pixels represent heavier load and darker pixels represent lighter load, which clearly shows the uneven load across pixel threads.

early cull by adaptive radius and axis-aligned bounding box can accelerate *Render* stage by reducing the number of serial Gaussian, the rendering efficiency is still limited by the *uneven load (computational time)* across pixel threads (an example shown in Fig. 4):

1) Uneven load across tiles: Since a 3D scene usually has different frequency information in different regions, where regions with higher frequency requires more Gaussians, the number of Gaussian of each tile varies, leading to uneven load among different tiles.

2) Uneven load within tiles: Within a tile, for a certain Gaussian, different pixels have different splatting opacity due to different distances to Gaussian center. Since the front-to-back blending will be stopped when the accumulated opacity exceeds a threshold, the actual number of rendered Gaussian is different across pixels.

Based on the nature of GPU thread waiting, uneven load across pixel threads leads to lower rendering efficiency [Liu and Vinter 2015]. To address the uneven load issue, we propose a *load balancing* algorithm for Gaussian splatting. We first reduce the number of Gaussian to be rendered for heavy-load pixels, and to address the rendering quality decrease caused by fewer Gaussians, we further increase the number of Gaussian to be rendered for light-load pixels.

Pixel Loads Distribution. We first compute the distribution of load of each pixel thread, which is equivalent to the distribution of the number of Gaussian ℓ_i to be rendered for each pixel :

$$\ell_i = \sum_{j \in N} \mathbb{1}\left(G_j, p_i\right), \tag{12}$$

where N denotes the overall number of Gaussian and $\mathbb{1}\left(G_j, p_i\right)$ is an indicator representing whether Gaussian j contributes to pixel i.

Load Balancing Loss. Given that the rendering efficiency under uneven loads hinges on heavy-load threads, to improve speed, the loads for these pixel threads need to be reduced. Meanwhile, encouraging light-load threads to enhance their information by increasing Gaussians to be rendered will not compromise efficiency. Thus, our load balancing strategy aims to reduce the number of Gaussian to be rendered in heavy-load pixels, and increase that in light-load pixels. We adopt the standard deviation to estimate the difference of loads across pixels, and design the load balancing loss as follows:

$$\mathcal{L}_{load} = std_{i \in HW}\left(\ell_i\right), \tag{13}$$

where H,W denotes the screen size and std represents standard deviation function. The smaller std indicates more balanced load distribution, and smaller difference in the number of Gaussian processed serially by each pixel thread, thus higher rendering efficiency.

Total Loss. To balance synthesis quality and rendering speed, we incorporate the load balancing loss with the L1-Loss and SSIM-Loss into the overall loss function, and the total loss is formulated as:

$$\mathcal{L} = \lambda_{L1}\mathcal{L}_1 + \lambda_{ssim}\mathcal{L}_{ssim} + \lambda_{load}\mathcal{L}_{load}, \tag{14}$$

where λ_{L1}, λ_{ssim}, λ_{load} are the weight coefficients of the L1 loss, SSIM loss, and load balancing loss, respectively. The numerical values satisfy $\lambda_{L1} + \lambda_{ssim} + \lambda_{load} = 1$.

5 Experiments

5.1 Experimental Settings

5.1.1 Datasets and Metrics. To validate the effectiveness of our method, we utilize the same datasets and metrics as those used in 3DGS [Kerbl et al. 2023]. Specifically, the datasets comprises all scenes from Mip-NeRF360 [Barron et al. 2022], two scenes from Tanks&Temples [Knapitsch et al. 2017], and two scenes from Deep-Blending [Hedman et al. 2018], encompassing both bounded indoor scenes and unbounded outdoor environments. The evaluation metrics include training duration, model size, rendering speed measured in frames per second (FPS), and synthesis quality assessed using PSNR, SSIM [Wang et al. 2004], and LPIPS [Zhang et al. 2018]. We employ identical hyperparameter settings across all experiments, with results reported on a single NVIDIA RTX 3090 GPU.

5.1.2 Implementation. Since adaptive radius and load acquisition in our method are based on Gaussian rasterization, these core methods are implemented using CUDA kernels, while load assessment and loss adjustments are handled by PyTorch. For bounding circle with adaptive radius, as the larger eigenvalue $\lambda^{'}_{max}$ of the projected covariance $\Sigma^{'}$ is already computed, the only thing we need to do is to multiply it by the opacity coefficient $2ln\left(\frac{\sigma_i}{\alpha_{low}}\right)$ instead of 3 to yield the adaptive radius r_{ad} as in Eq. (7). Note that we maintain a minimum splatting opacity α_{low} of $\frac{1}{255}$ for optimal rendering quality, though higher values could further accelerate rendering. Regarding the load balancing approach, to prioritize quality optimization while accelerating rendering, the weight of its loss term λ_{load} is set to 0.45, while the sum weight of remaining loss is reduced to 0.55.

5.2 Comparisons

We compare *AdR-Gaussian* (**Ours-Full** denotes our full method, and **Ours-AABB** denotes AABB-only method) to 3DGS [Kerbl et al. 2023], and state-of-the-art 3DGS faster rendering methods: Light-Gaussian [Fan et al. 2023], C3DGS [Lee et al. 2023], EAGLES [Girish et al. 2023], and GES [Hamdi et al. 2024]. The qualitative and quantitative comparisons are shown in Fig. 5 and Table 1, respectively.

Despite varying amounts and distributions of information across datasets, our *AdR-Gaussian* with only AABB for 3DGS (denoted as **Ours-AABB**) achieves higher rendering efficiency than any existing method on every dataset. Specifically, early culling with AABB has significantly improved the rendering speed to 185% of the baseline (3DGS), with an average rendering speed of 401 FPS on the three test datasets. Moreover, experiments demonstrate *Ours-AABB* outperforms comparison methods in rendering on Mip-NeRF360

Table 1. Quantitative evaluation of our method compared to the latest works, computed over the same full dataset as 3DGS. The bold results represent the best performance, while the underlined results stand for the second-best among all the results obtained in our own experiments. *Ours-AABB* shows the results for our method with only the axis-aligned bounding box for Gaussian splatting.

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Method|Metric	Train↓	FPS↑	Mem↓	SSIM↑	PSNR↑	LPIPS↓	Train↓	FPS↑	Mem↓	SSIM↑	PSNR↑	LPIPS↓	Train↓	FPS↑	Mem↓	SSIM↑	PSNR↑	LPIPS↓
3DGS	29m15s	185.85	780.7MB	0.813	27.49	0.220	14m56s	253.39	437.5MB	0.844	23.64	**0.178**	24m14s	213.00	677.5MB	0.899	29.50	**0.246**
LightGaussian	34m30s	212.32	**58.7MB**	0.803	27.03	0.238	17m49s	341.30	**33.9MB**	0.834	23.41	0.196	28m19s	259.15	**52.0MB**	0.896	29.12	0.258
C3DGS	37m5s	205.22	122.7MB	0.797	26.99	0.247	19m8s	301.95	93.0MB	0.831	23.42	0.202	28m31s	310.19	104.0MB	0.902	29.84	0.257
EAGLES	21m55s	177.87	68.11MB	0.805	27.14	0.239	11m22s	300.68	34.0MB	0.836	23.34	0.200	19m41s	195.64	61.5MB	**0.907**	**29.95**	0.249
GES	21m50s	241.67	332.9MB	0.792	26.95	0.258	10m56s	339.22	214.5MB	0.836	23.36	0.197	19m04s	279.93	367.5MB	0.901	29.60	0.252
Ours-AABB	27m49s	336.15	786.7MB	**0.814**	**27.51**	**0.219**	14m3s	419.07	445.5MB	**0.846**	**23.76**	**0.178**	22m51s	449.21	693.0MB	0.899	29.50	**0.246**
Ours-Full	**17m44s**	**589.61**	274.3MB	0.783	26.90	0.272	**9m2s**	**718.42**	192.0MB	0.832	23.53	0.205	**15m32s**	**716.46**	335.5MB	0.901	29.65	0.254

Table 2. Ablation on same full dataset as comparisons. The experimentation begins with the original 3DGS project and sequentially introduces each sub method of ours. The bold results represent the best performance among all the results obtained in our ablation experiments.

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Method|Metric	Train↓	FPS↑	Mem↓	SSIM↑	PSNR↑	LPIPS↓	Train↓	FPS↑	Mem↓	SSIM↑	PSNR↑	LPIPS↓	Train↓	FPS↑	Mem↓	SSIM↑	PSNR↑	LPIPS↓
3DGS	29m15s	185.85	780.7MB	0.813	27.49	0.220	14m56s	253.39	437.5MB	0.844	23.64	0.178	24m14s	213.00	677.5MB	0.899	29.50	**0.246**
3DGS+Adaptive radius	28m38s	221.90	782.2MB	0.813	27.50	0.220	14m25s	361.00	439.5MB	0.845	23.68	**0.177**	23m35s	320.10	678.5MB	0.899	29.51	**0.246**
3DGS+AABB	27m49s	336.15	786.7MB	**0.814**	**27.51**	**0.219**	14m3s	419.07	445.5MB	**0.846**	**23.76**	0.178	22m51s	449.21	693.0MB	0.899	29.50	**0.246**
3DGS+AABB+Load balance (Ours)	**17m44s**	**589.61**	**274.3MB**	0.783	26.90	0.272	**9m2s**	**718.42**	**192.0MB**	0.832	23.53	0.205	**15m32s**	**716.46**	**335.5MB**	**0.901**	**29.65**	0.254

Table 3. Experiments result on the same three dataset for the combination of *Ours-Full* and the pruning startegy in LightGaussian.

Dataset	Mip-NeRF360		Tanks&Temples		Deep Blending	
Method|Metric	FPS↑	PSNR↑	FPS↑	PSNR↑	FPS↑	PSNR↑
Ours-Full	589.61	26.90	718.42	23.53	716.46	29.65
(+)Pruning	741.12	26.81	1109.70	23.42	1035.37	29.61

and Tanks&Temples datasets. Compared to the baseline 3DGS, we achieve lossless acceleration, which is due to we only early cull Gaussian-Tile pairs that contribute nothing to the final color.

Additionally, our full *AdR-Gaussian* method with both early culling and load balancing (denoted as **Ours-Full**) achieves the highest rendering speed among all methods, 310% on average compared to 3DGS. This is realized by the reduction in the number of Gaussian to be rendered for heavy-load pixels, *i.e.* reduction in thread waiting, which also improves our training speed and model size on every dataset compared to the baseline. Meanwhile, *Ours-Full* can achieve superior quality on *Deep Blending* dataset, because our load balancing also increases the number of Gaussian to be rendered for light-load pixels, which enhances the information on these pixels.

5.3 Ablation Studies

To clearly assess the performance of each component of our methods, we conduct ablation studies as follows: we begin with the original 3DGS project, and sequentially add each module: early culling with adaptive radius, and with AABB for 3DGS, and load balancing. The quantitative results are presented in Table 2, where the contribution of each module can be obtained by comparing corresponding rows.

Adaptive Radius. The ablation experiment shows that early-culling with adaptive radius enhances rendering speed for all test datasets, without rendering quality loss, and improving training time. Adaptive radius achieves rendering acceleration by moving part of serial culling from the Gaussian-serial *Render* stage earlier into the Gaussian-parallel *Preprocess* stage. Moreover, this method does not sacrifice rendering quality, achieving slight rendering quality improvement while largly improving speed.

Axis-aligned Bounding Box. Experiments on three datasets demonstrate that early culling with axis-aligned bounding box for Gaussian splatting achieves higher FPS, *i.e.,* higher rendering efficiency. Since early-culling with AABB can effectively cull the Gaussian-Tile pairs in the minor-axis direction, which can not be achieved by the bounding circle calculated by adaptive radius, early-culling with AABB moves more serial culling in *Render* stage earlier into parallel culling in *Preprocess* stage, leading to further acceleration. Experiments show that such early culling methods are lossless and suitable for any scene.

Load Balancing. Experimental results show load balancing method can further improve speed and multiple metrics despite little sacrifice of quality. Load balancing strategy continuously reduces the the number of Gaussian to be rendered for heavy-load pixels with load balancing loss, which enables a comprehensive optimization of rendering speed and other metrics. Additionally, load balancing also enriches information for light-load pixels, which can offset the quality loss. Regarding the application of load balancing in different scenes, it is more suitable for scenes with less high-frequency information, such as the scenes in the *Deep Blending* dataset. The overall acceleration ratio increases with higher information frequency, sacrificing more quality, while local areas with relatively low-frequency information can always achieve more accurate modeling, hedging against the quality loss.

5.4 Discussions

5.4.1 Limitation. The adjustment of minimum splatting opacity is not fully compatible with the training of 3D Gaussians. Since Gaussian opacity is reinitialized every fixed intervals, setting an splatting opacity threshold higher than this initialized value can result in the culling of all Gaussians, leading to rendering failures. Meanwhile, when only rendering is required, raising the splatting opacity threshold can safely speed up the process.

5.4.2 Combination. Our method focuses on the rasterization rather than the representation of 3D Gaussian, allowing for easy integration with other 3DGS methods, including those aimed at accelerating rendering. As shown in Table 3, by adding the pruning strategy in LightGaussian, the rendering speed of *Ours-Full* can be further accelerated with only a minor degeneration of quality.

6 Conclusion

In this paper, we propose *AdR-Gaussian*, an acceleration for 3D Gaussian Splatting, which achieves a rendering speed of 310% while maintaining equivalent or even better quality than its baseline (3DGS). By moving part of serial culling in *Render* stage earlier into parallel culling in *Preprocess* stage, and balances the load across different pixel threads to minimum thread waiting, *AdR-Gaussian* is able to significantly accelerate the rendering speed of Gaussian Splatting. Compared with state-of-the-art fast rendering methods for 3DGS, our AdR-Gaussian shows much higher rendering efficiency.

Acknowledgments

We thank the reviewers for their constructive comments. This work was supported by National Natural Science Foundation of China (No. 72192821, 62302296, 62302297, 62272447), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), Shanghai Sailing Program (22YF1420300), Young Elite Scientists Sponsorship Program by CAST (2022QNRC001), the Fundamental Research Funds for the Central Universities (project number: YG2023QNA35, YG2023QNB17, YG2024QNA44).

References

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 5460–5469. https://doi.org/10.1109/CVPR52688.2022.00539

Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. 2001. Unstructured lumigraph rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001, Los Angeles, California, USA, August 12-17, 2001*. ACM, 425–432. https://doi.org/10.1145/383259.383309

Junyi Cao, Zhichao Li, Naiyan Wang, and Chao Ma. 2024. Lightning NeRF: Efficient Hybrid Scene Representation for Autonomous Driving. *CoRR* abs/2403.05907 (2024), 1–7. https://doi.org/10.48550/ARXIV.2403.05907 arXiv:2403.05907

Guikun Chen and Wenguan Wang. 2024. A Survey on 3D Gaussian Splatting. *CoRR* abs/2401.03890 (2024), 1–19. https://doi.org/10.48550/ARXIV.2401.03890 arXiv:2401.03890

Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yikai Wang, Zhongang Cai, Lei Yang, Huaping Liu, and Guosheng Lin. 2023a. GaussianEditor: Swift and Controllable 3D Editing with Gaussian Splatting. *CoRR* abs/2311.14521 (2023), 1–14. https://doi.org/10.48550/ARXIV.2311.14521 arXiv:2311.14521

Zhiqin Chen, Thomas A. Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023b. MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 16569–16578. https://doi.org/10.1109/CVPR52729.2023.01590

Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. 1996. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*. ACM, 11–20. https://doi.org/10.1145/237170.237191

Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2023. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. *CoRR* abs/2311.17245 (2023), 1–16. https://doi.org/10.48550/ARXIV.2311.17245 arXiv:2311.17245

John Flynn, Michael Broxton, Paul E. Debevec, Matthew DuVall, Graham Fyffe, Ryan S. Overbeck, Noah Snavely, and Richard Tucker. 2019. DeepView: View Synthesis With Learned Gradient Descent. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2367–2376. https://doi.org/10.1109/CVPR.2019.00247

Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. 2023. EAGLES: Efficient Accelerated 3D Gaussians with Lightweight EncodingS. *CoRR* abs/2312.04564 (2023), 1–13. https://doi.org/10.48550/ARXIV.2312.04564 arXiv:2312.04564

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. 1996. The Lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*. ACM, 43–54. https://doi.org/10.1145/237170.237200

Abdullah Hamdi, Luke Melas-Kyriazi, Guocheng Qian, Jinjie Mai, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. 2024. GES: Generalized Exponential Splatting for Efficient Radiance Field Rendering. *CoRR* abs/2402.10128 (2024), 1–39. https://doi.org/10.48550/ARXIV.2402.10128 arXiv:2402.10128

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel J. Brostow. 2018. Deep blending for free-viewpoint image-based rendering. *ACM Trans. Graph.* 37, 6 (2018), 257. https://doi.org/10.1145/3272127.3275084

Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul E. Debevec. 2021. Baking Neural Radiance Fields for Real-Time View Synthesis. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 5855–5864. https://doi.org/10.1109/ICCV48922.2021.00582

Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. 2023. Tri-MipRF: Tri-Mip Representation for Efficient Anti-Aliasing Neural Radiance Fields. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 19717–19726. https://doi.org/10.1109/ICCV51070.2023.01811

Chiyu Max Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas A. Funkhouser. 2020. Local Implicit Grid Representations for 3D Scenes. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 6000–6009. https://doi.org/10.1109/CVPR42600.2020.00604

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4 (2023), 139:1–139:14. https://doi.org/10.1145/3592433

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and temples: benchmarking large-scale scene reconstruction. *ACM Trans. Graph.* 36, 4 (2017), 78:1–78:13. https://doi.org/10.1145/3072959.3073599

Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. 2021. Point-Based Neural Rendering with Per-View Optimization. *Comput. Graph. Forum* 40, 4 (2021), 29–43. https://doi.org/10.1111/CGF.14339

Christoph Lassner and Michael Zollhöfer. 2021. Pulsar: Efficient Sphere-Based Neural Rendering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 1440–1449. https://doi.org/10.1109/CVPR46437.2021.00149

Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. 2023. Compact 3D Gaussian Representation for Radiance Field. *CoRR* abs/2311.13681 (2023), 1–13. https://doi.org/10.48550/ARXIV.2311.13681 arXiv:2311.13681

Marc Levoy and Pat Hanrahan. 1996. Light Field Rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*. ACM, 31–42. https://doi.org/10.1145/237170.237199

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. https://proceedings.neurips.cc/paper/2020/hash/b4b758962f17808746e9bb832a6fa4b8-Abstract.html

Weifeng Liu and Brian Vinter. 2015. Speculative segmented sum for sparse matrix-vector multiplication on heterogeneous processors. *Parallel Comput.* 49 (2015), 179–193.

Hidenobu Matsuki, Riku Murai, Paul H. J. Kelly, and Andrew J. Davison. 2023. Gaussian Splatting SLAM. *CoRR* abs/2312.06741 (2023), 1–21. https://doi.org/10.48550/ARXIV.2312.06741 arXiv:2312.06741

Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local light field fusion: practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.* 38, 4 (2019), 29:1–29:14. https://doi.org/10.1145/3306346.3322980

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: representing scenes as neural radiance fields for view synthesis. *Communication of ACM* 65, 1 (dec 2021), 99–106.

Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. 2024. Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis. *CoRR* abs/2401.02436 (2024), 1–16. https://doi.org/10.48550/ARXIV.2401.02436 arXiv:2401.02436

Michael Niemeyer, Lars M. Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without

3D Supervision. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 3501–3512. https://doi.org/10.1109/CVPR42600.2020.00356

Jeong Joon Park, Peter R. Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 165–174. https://doi.org/10.1109/CVPR.2019.00025

Shenhan Qian, Tobias Kirschstein, Liam Schoneveld, Davide Davoli, Simon Giebenhain, and Matthias Nießner. 2023. GaussianAvatars: Photorealistic Head Avatars with Rigged 3D Gaussians. *CoRR* abs/2312.02069 (2023), 1–13. https://doi.org/10.48550/ARXIV.2312.02069 arXiv:2312.02069

Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. 2021. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 14315–14325. https://doi.org/10.1109/ICCV48922.2021.01407

Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T. Barron, and Peter Hedman. 2023. MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. *ACM Trans. Graph.* 42, 4 (2023), 89:1–89:12. https://doi.org/10.1145/3592426

Zixi Shu, Ran Yi, Yuqi Meng, Yutong Wu, and Lizhuang Ma. 2023. RT-Octree: Accelerate PlenOctree Rendering with Batched Regular Tracking and Neural Denoising for Real-time Neural Radiance Fields. In *SIGGRAPH Asia 2023 Conference Papers, SA 2023, Sydney, NSW, Australia, December 12-15, 2023*. ACM, 99:1–99:11. https://doi.org/10.1145/3610548.3618214

Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 1119–1130. https://proceedings.neurips.cc/paper/2019/hash/b5dc4e5d9b495d0196f61d45b26ef33e-Abstract.html

Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. 2023. Splatter Image: Ultra-Fast Single-View 3D Reconstruction. *CoRR* abs/2312.13150 (2023), 1–16. https://doi.org/10.48550/ARXIV.2312.13150 arXiv:2312.13150

Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. 2023. DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation. *CoRR* abs/2309.16653 (2023), 1–18. https://doi.org/10.48550/ARXIV.2309.16653 arXiv:2309.16653

Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul P. Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, Tomas Simon, Christian Theobalt, Matthias Nießner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhöfer, and Vladislav Golyanik. 2022. Advances in Neural Rendering. *Comput. Graph. Forum* 41, 2 (2022), 703–735. https://doi.org/10.1111/CGF.14507

Tengfei Wang, Bo Zhang, Ting Zhang, Shuyang Gu, Jianmin Bao, Tadas Baltrusaitis, Jingjing Shen, Dong Chen, Fang Wen, Qifeng Chen, and Baining Guo. 2023. RODIN: A Generative Model for Sculpting 3D Digital Avatars Using Diffusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 4563–4573. https://doi.org/10.1109/CVPR52729.2023.00443

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. https://doi.org/10.1109/TIP.2003.819861

Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 2023. 4D Gaussian Splatting for Real-Time Dynamic Scene Rendering. *CoRR* abs/2310.08528 (2023), 1–14. https://doi.org/10.48550/ARXIV.2310.08528 arXiv:2310.08528

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 5732–5741. https://doi.org/10.1109/ICCV48922.2021.00570

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2023. Mip-Splatting: Alias-free 3D Gaussian Splatting. *CoRR* abs/2311.16493 (2023), 1–19. https://doi.org/10.48550/ARXIV.2311.16493 arXiv:2311.16493

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 586–595. https://doi.org/10.1109/CVPR.2018.00068

Hongyu Zhou, Jiahao Shao, Lu Xu, Dongfeng Bai, Weichao Qiu, Bingbing Liu, Yue Wang, Andreas Geiger, and Yiyi Liao. 2024. HUGS: Holistic Urban 3D Scene Understanding via Gaussian Splatting. *CoRR* abs/2403.12722 (2024), 1–18. https://doi.org/10.48550/ARXIV.2403.12722 arXiv:2403.12722

M. Zwicker, H. Pfister, J. van Baar, and M. Gross. 2002. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics* 8, 3 (2002), 223–238. https://doi.org/10.1109/TVCG.2002.1021576

Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus H. Gross. 2001. EWA Volume Splatting. In *12th IEEE Visualization Conference, IEEE Vis 2001, San Diego, CA, USA, October 24-26, 2001, Proceedings*. IEEE Computer Society, 29–36. https://doi.org/10.1109/VISUAL.2001.964490

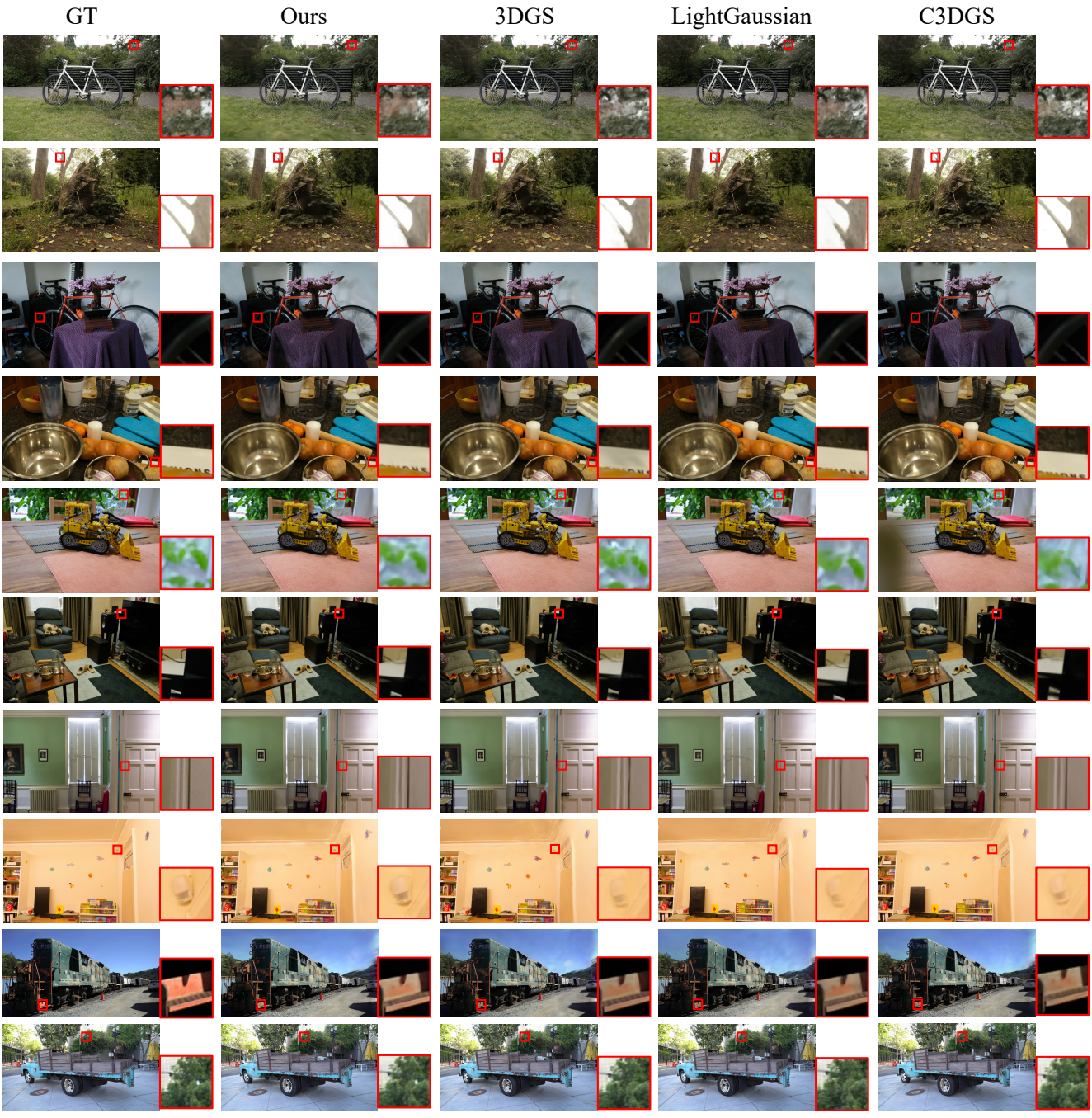

Fig. 5. Our qualitative comparison with 3DGS and recent accelerated rendering methods, where images are from held-out test views. The scenes are, from the top down: *BICYCLE*, *STUMP*, *BONSAI*, *COUNTER*, *KITCHEN*, and *ROOM* from the Mip-NeRF360 dataset; *DRJOHNSON* and *PLAYROOM* from the Deep Blending dataset; *TRAIN* and *TRUCK* from Tanks&Temples dataset. Detailed comparisons are highlighted with red rectangles, where the larger rectangle is the zoomed-in view of the smaller one.