SEELE: A Unified Acceleration Framework for Real-Time Gaussian Splatting on Mobile Devices

He Zhu[1*] Xiaotong Huang[1*] Zihan Liu[12] Weikai Lin[3] Xiaohong Liu[1] Zhezhi He[1]
Jingwen Leng[12] Minyi Guo[12] Yu Feng[1†]
[1]Shanghai Jiao Tong University [2]Shanghai Qi Zhi Institute [3]University of Rochester
{zhcon16, hxt0512, y-feng}@sjtu.edu.cn
Project Page: http://seele-project.netlify.app

Abstract

3D Gaussian Splatting (3DGS) has become a crucial rendering technique for many real-time applications. However, the limited hardware resources on today's mobile platforms hinder these applications, as they struggle to achieve real-time performance. In this paper, we propose SEELE*, a general framework designed to accelerate the 3DGS pipeline for resource-constrained mobile devices.*

Specifically, we propose two GPU-oriented techniques: hybrid preprocessing and contribution-aware rasterization. Hybrid preprocessing alleviates the GPU compute and memory pressure by reducing the number of irrelevant Gaussians during rendering. The key is to combine our view-dependent scene representation with online filtering. Meanwhile, contribution-aware rasterization improves the GPU utilization at the rasterization stage by prioritizing Gaussians with high contributions while reducing computations for those with low contributions. Both techniques can be seamlessly integrated into existing 3DGS pipelines with minimal fine-tuning. Collectively, our framework achieves up to 6.3× speedup and 39.1% model reduction while achieving superior rendering quality compared to existing methods. Our codes will be released upon publication.

1. Introduction

3D Gaussian Splatting (3DGS) [25] has emerged as a vital technique in many rendering-related domains, including autonomous driving [26, 36, 37, 47], augmented and virtual reality (AR/VR) [7, 21, 24, 45]. These real-time applications require high rendering performance to ensure seamless quality of services. However, the limited computational resources of mobile platforms often constrain these applications, preventing them from reaching their full potential.

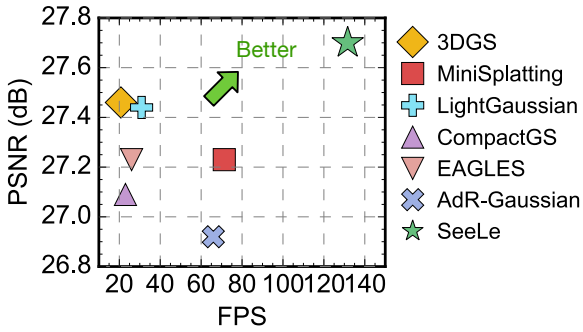


Fig. 1. Our acceleration framework, SEELE, achieves up to 6.3× speedup against the state-of-the-art 3DGS algorithms.

For instance, Nvidia AGX Orin, a leading-edge compute module for modern vehicles, offers only 3.4% of the computational resources of a Nvidia A100 workstation GPU. On Nvidia's AGX Orin, 3DGS [25] barely achieves 20 frames-per-second (FPS) on real-world datasets [4, 20, 28], far from the real-time requirement, i.e., 90 FPS of VR [48]. An interesting challenge is to achieve real-time on devices with only 3-4% of the resources compared to high-end GPUs.

To achieve this goal, we must understand the major performance bottlenecks in the 3DGS rendering pipelines. We summarized the bottlenecks into three aspects: computational intensity, rendering inefficiency, and memory budget. This paper proposes a *one-stop solution* framework, SEELE [*], that addresses the inefficiencies across these three dimensions (Sec. 3). Next, we briefly discuss the inefficiencies in today's 3DGS rendering pipeline and highlight how our proposed solution addresses these limitations.

Computational Intensity. One primary challenge in the 3DGS pipeline is its computational intensity. From

*Equal contribution.
†Corresponding author.

*The name is an organization that comes from a famous Japanese anime, *Neon Genesis Evangelion*. This organization's mission is to accomplish the Human Instrumentality Project. Similarly, the goal of our project is to resolve all the performance bottlenecks in 3DGS.

an individual pixel's perspective, rendering one pixel typically involves processing thousands of Gaussian points, with each GPU thread rendering one pixel. This rendering process imposes substantial computational overhead on resource-constrained devices. Prior solutions [9, 10] primarily focus on reducing the overall model size, namely the number of Gaussian points, but these methods often trade off rendering quality for performance.

In contrast, we propose *hybrid preprocessing* with a view-dependent scene representation that dynamically loads the relevant Gaussians to GPU memory during runtime (Sec. 3.3). Our data representation naturally identifies the necessary Gaussians contributing to the current viewpoint while leaving other irrelevant Gaussians untouched. On average, this method alone outperforms the state-of-the-art rendering algorithms while achieving a 2.8× speedup.

Rendering Inefficiency. The second issue with the current 3DGS rendering pipeline is that all splatted Gaussians go through the same rasterization pipeline regardless of their contribution to the final pixels. Our experiment shows that, for each pixel, 1.5% of the Gaussians contribute to 99% of the final pixel. Therefore, we argue that this uniform treatment to all Gaussians leads to severe rendering inefficiencies, as *less significant Gaussians should naturally receive fewer computational budgets*. However, prior work [14, 31, 34], such as pruning or quantization techniques, still retains a uniform rendering pipeline and fails to differentiate Gaussians based on their contributions.

On the contrary, we propose contribution-aware rasterization from a pixel-centric perspective (Sec. 3.4). Rather than reducing computation uniformly, we dynamically identify and prioritize Gaussians with high contributions and reduce computation budgets for those with low contributions. Gaussians with low contributions, which typically contribute to low-frequency textures (as shown in Fig. 4), are allocated less computation by dynamically skipping insignificant operations. We show that this method achieves an additional 1.4× speedup while retaining the same rendering quality.

Memory Budget. Our last contribution eases the peak GPU memory. As model complexity scales up, the GPU memory requirements often become unsustainable due to the increasing number of Gaussians involved. Solutions [31, 44] apply offline compression techniques to reduce memory usage, but they overlook a key opportunity: rendering a given viewpoint does not require loading all Gaussians into GPU memory simultaneously.

Leveraging this key observation, we design proactive memory management that asynchronously loads necessary data to GPU memory without disrupting the critical path of rendering. Combined with our view-dependent scene representation, our memory management significantly reduces peak GPU memory usage. On average, our approach achieves a 39.1% reduction in model size, allowing large-scale rendering on mobile devices.

Collectively, our framework achieves up to 6.3× speedup and 39.1% runtime model reductions, all while retaining a better rendering quality compared to state-of-the-art algorithms. Furthermore, our techniques are orthogonal to existing optimizations and can be seamlessly integrated with other mainstream Gaussian splatting pipelines.

The contribution of this paper is summarized as follows:

- We introduce a view-dependent scene representation that combines online and offline filtering to reduce computation overhead and runtime GPU memory.
- We propose a contribution-aware rasterization algorithm that dynamically skips the insignificant computations and improves the parallel efficiency.
- We design an integrated co-training procedure that integrates the aforementioned techniques and achieves better rendering quality against the corresponding baselines.

2. Related Work

2.1. Efficient Data Representation

A 3DGS model is essentially a set of points in 3D space that represent the physical structure of the world. The model size often becomes one of the key bottlenecks in rendering performance and memory usage. To reduce the model size, several studies have proposed various data representations to optimize storage and performance. Some [5, 6, 8, 40–42] reduce GPU memory usage by leveraging encoding techniques to compress the model. For example, CompactGS [31] employs vector quantization to compress the model offline and decodes it at runtime using a pretrained codebook. Similarly, EAGLES [14] uses an encoding network to compress the neural radiance field into a compact representation. However, both methods introduce non-trivial execution overhead during runtime, which limits their applicability in computation-constrained scenarios.

Other pruning-based approaches use a learnable mask [38, 46, 51] or a significance score [9–11, 19, 39] to eliminate redundant Gaussians. However, these methods have to make a trade-off between rendering quality and efficiency. Some studies adopt structured data representations such as octrees [44] and kd-tree [26] to organize Gaussian points while some researches [17, 22, 32, 50] explore alternative primitives for more efficient scene representation. However, they introduce additional runtime overhead in loading or rasterizing Gaussians.

In contrast to prior studies, we focus on real-time performance. Our view-dependent scene representation achieves significant model size reduction with minimal runtime overhead and no quality compromise.

2.2. Rasterization Optimization

In addition to pruning Gaussian points, another line of research focuses on optimizing the performance of the 3DGS pipeline itself. Recent studies propose various online filtering techniques, such as axis-aligned bounding box (AABB) [18, 27, 49] and oriented bounding box (OBB) [15, 29] intersection tests, to perform more fine-grained filtering before rasterization. These works aim to reduce the number of Gaussians processed during the rasterization stage at runtime, thereby reducing the computational workload.

On the other hand, some methods [12, 16, 33] focus on optimizing the rasterization stage. For example, FlashGS [12] designs a software pipelining to overlap data fetching with rasterization, improving overall efficiency. In contrast, Balanced3DGS [16] addresses workload imbalance by introducing an offline scheduling mechanism to rebalance workloads at both the block and tile levels. Additionally, a few studies [29, 30, 35, 43] explore the new hardware designs to support 3DGS, further advancing the efficiency and scalability of the pipeline.

Unlike prior works, our optimization takes a unique angle to accelerate 3DGS by addressing the inherent contribution imbalance across Gaussian points. Our method dynamically assigns computational budgets to Gaussians based on their contributions to the final rendered image. Moreover, our algorithm is naturally GPU-friendly and alleviates execution inefficiencies such as warp divergence, achieving better overall performance with no hardware modifications.

3. Methodology

3.1. Preliminaries

3DGS algorithms adhere to an explicit point-based representation, where each point is modeled by a Gaussian ellipsoid (a.k.a., Gaussian for short) that captures the geometric and textural properties of the scene. The geometric property of a Gaussian is governed by its 3D centroid position x and a 3D covariance matrix Σ. The textural properties are described by two components: its opacity o and spherical harmonic (SH) coefficients. Using these properties, the 3DGS pipeline renders a frame on a tile-by-tile basis via three steps: *preprocessing*, *sorting*, and *rasterization*.

Preprocessing. Given a camera pose and a view direction, the preprocessing stage first filters out Gaussians outside the current view frustum. Meanwhile, this stage identifies the intersection between Gaussians and rendering tiles.

Sorting. Once each tile collects its intersected Gaussians, the sorting stage determines the rendering order of those Gaussians. This stage ensures that all points are rendered, from the closest to the furthest, based on their depth.

Rasterization. Once all Gaussians are sorted, the rasterization stage renders these Gaussians tile-by-tile. Within each tile, every pixel iterates through the same set of Gaussians, computing transparency and accumulating their colors into the pixel in the sorted order. Eqn. 1 governs the color accumulation process of pixel $\mathbf{p}$,

$$C(\mathbf{p}) = \sum_{i=1}^{N} \Gamma_i \alpha_i \mathbf{c}_i, \text{ where } \Gamma_i = \prod_{j=1}^{i-1}(1-\alpha_j), \quad (1)$$

where Γ_i denotes the accumulative transmittance of pixel $\mathbf{p}$ from the Gaussian 1st to the $i-1$th. α_i and $\mathbf{c}_i$ stand for the transparency and the color at the ith Gaussian, respectively. The transparency of a Gaussian α_i is calculated by the Gaussian opacity o_i, the 2D position x' and the 2D covariance matrix, Σ' [52],

$$\alpha_i = o_i e^{-\frac{1}{2}(\mathbf{p}-x')^T \Sigma'^{-1}(\mathbf{p}-x')}. \quad (2)$$

$\mathbf{c}_i$ is a function of the Gaussian spherical harmonics.

Note that, if the Gaussian's α_i value is insignificant (less than α_θ, i.e., $\frac{1}{255}$), this Gaussian will be skipped in the color accumulation to avoid numerical instabilities. The color accumulation process terminates once the cumulative transmittance Γ_i surpasses a predefined threshold, Γ_θ.

3.2. Overview

Fig. 2 gives the overview of our proposed acceleration framework. To address three rendering inefficiencies highlighted in Sec. 1, we introduce two key components: *hybrid preprocessing* and *contribution-aware rasterization* to reduce the runtime computation and peak memory.

Our hybrid preprocessing combines the merits of both offline and online techniques. During the offline processing, we design a view-dependent scene representation that clusters Gaussians based on their contributions to a set of closely related rendering viewpoints. At runtime, our framework can asynchronously prefetch the relevant clusters into the GPU memory before rendering. Our view-dependent scene representation inherently eliminates Gaussians that are irrelevant to the current rendering while within the view frustum. Combined with online filtering, we further reduce the number of Gaussians intersected for each tile in the subsequent stages.

Meanwhile, we propose a novel contribution-aware rasterization to accelerate the rasterization pipeline itself. The key idea of our approach is to penalize the computations assigned to insignificant Gaussians purposely, allowing for efficient rendering without compromising quality.

Specifically, our rasterization pipeline organizes pixels into small groups while still keeping one thread responsible for one pixel. At runtime, only a single pixel within each pixel group calculates its Gaussian transparency α rather than having all pixels compute α. The purpose is to let this pixel identify Gaussians that potentially contribute less to the pixel group, so we can exclude these Gaussians from further rasterization. In this way, we eliminate insignificant

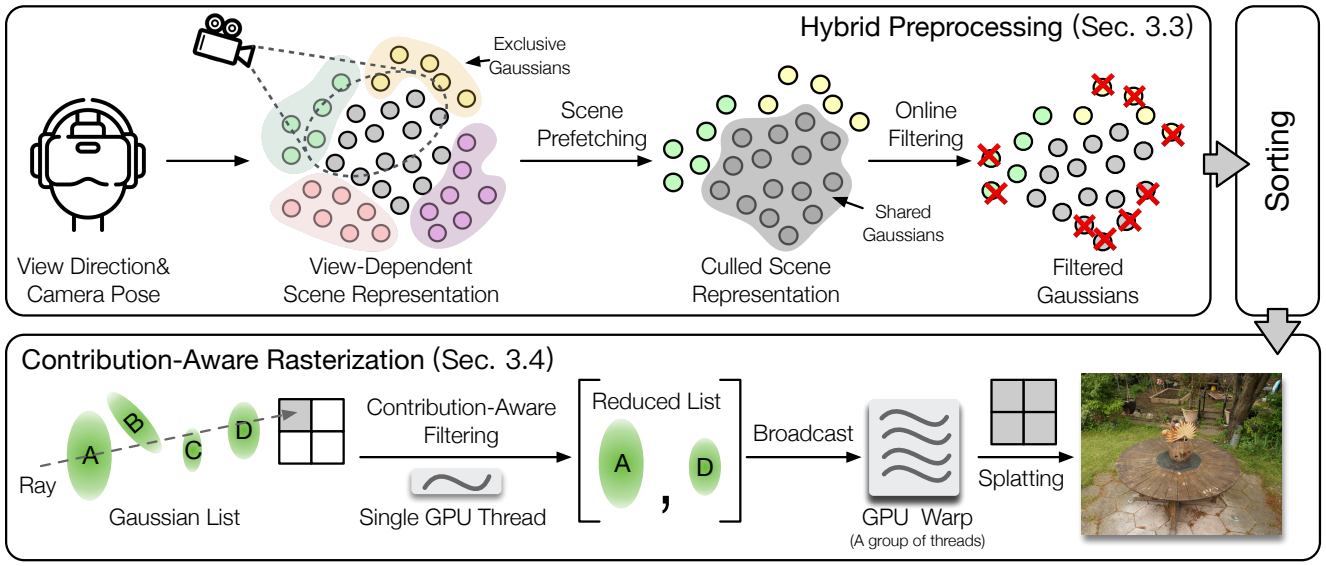
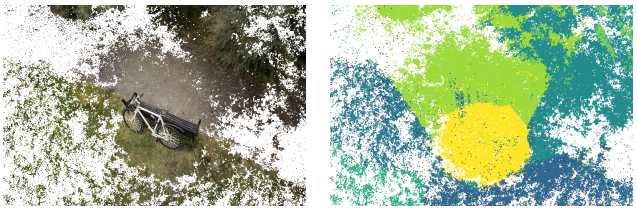


Fig. 2. The overview of SEELE. we modify the two steps, *preprocessing* and *rasterization*, and propose two novel techniques: *hybrid preprocessing* and *contribution-aware rasterization*, in Gaussian splatting. Hybrid preprocessing leverages offline coarse-grained scene clustering and online filtering to reduce the number of Gaussians before rasterization. Contribution-aware rasterization dynamically identifies insignificant Gaussians and skips them to accelerate the overall rendering pipeline.



(a) Original Gaussian points. (b) Clusters w/ shared points.

Fig. 3. Our scene representation clusters Gaussians into shared ones and exclusive ones. Here, we show the Gaussian positions *without* scales. The yellow points in Fig. 3b represent the shared Gaussians, while the other colors correspond to the exclusive Gaussians in different clusters.

Gaussian accumulation at runtime. Combined with our integrated co-training, we can achieve a smoother rendering experience. A more rigorous explanation of this approach is provided in Sec. 3.4.

3.3. Hybrid Preprocessing, HP

View-Dependent Scene Representation. The idea of this data representation is to preprocess Gaussians offline, thereby reducing computational effort during the online preprocessing stage. Our key observation is that different viewpoints rely on distinct sets of Gaussians for rendering. Viewpoints in close proximity are likely to share similar Gaussians, while those farther apart share fewer.

Based on this observation, our data representation classifies Gaussians into two categories: shared Gaussians and exclusive Gaussians, as shown in Fig. 3. Shared Gaussians are shared across all viewpoints and are frequently used for rendering frames from any viewpoint (yellow points in Fig. 3b). On the other hand, exclusive Gaussians are specifically attached to a group of viewpoints and are only used for rendering frames within that group (other colors in Fig. 3b).

Offline Clustering. The key to our approach lies in how to classify the Gaussians into shared and exclusive Gaussians. Initially, our method randomly samples camera poses and clusters them into N smaller groups based on the weighted similarity of their camera positions, $\overrightarrow{x}$ and view directions, $\overrightarrow{v}$. Specifically, we first normalize the camera positions, $\overrightarrow{x}$, then concatenate normalized $\overrightarrow{x}$ and $\overrightarrow{v}$ as a 1D dimensional vector, $(\overrightarrow{x}, \beta\overrightarrow{v})$, where β is a hyperparameter that balances the impact of position and orientation. Here, β is set to 1 for clustering all Gaussians.

For each cluster, we identify and reserve the Gaussians that are the primary contributors to pixel rendering. Specifically, for each pixel, we find the Top-k Gaussians based on their contribution to the cumulative transmittance, $\Gamma_i\alpha_i$, as the contribution metric. The rationale here is that the remaining Gaussians contribute minimally to the final pixel and can be safely discarded without compromising rendering quality. We choose a k value of 32, as we find that increasing k further yields minimal quality improvement.

Once we identify the top Gaussian contributors for each cluster, we find the union of top contributors across all clusters and grant these top Gaussians to be shared across all clusters. These granted Gaussians are then the shared Gaussians in our representation. The remaining Gaussians within each cluster are classified as exclusive Gaussians.

Algorithm. During the actual rendering, for a given

camera pose, our approach first identifies the nearest cluster based on the cluster centroid. Next, all Gaussians belonging to this cluster and its $M-1$ neighboring clusters are selected, as shown in Fig. 2. In this example, green points are the nearest cluster, and the yellow points are the nearest neighbors to that cluster (M is set to 2 in this toy example). The selected M clusters are used to render the current camera pose instead of all Gaussians. Note that, shared Gaussians, gray points in Fig. 2, are always stored in GPU memory since they are used for any poses. Unlike the original 3DGS, our approach retains only the relevant clusters to stay in GPU memory, thus reducing the peak GPU memory usage. Empirically, we set M to be 4, which strikes a good balance between performance and rendering quality.

Online Filtering. Once these Gaussians are loaded, they are transformed into screen space with a new 2D Gaussian position x' and covariance matrix Σ'. To filter unnecessary Gaussians for each tile, the intersection test typically uses 3σ envelope to approximate,

$$\sqrt{(\mathbf{p}-x_i')^T\Sigma_i'^{-1}(\mathbf{p}-x_i')}=3, \tag{3}$$

where $\mathbf{p}$ is the pixel position. Due to the redundancy of this approximation [18, 43], we further include the opacity o_i into our online filtering equation as follows. As we would skip insignificant α_i, which value is less than α_θ, $\frac{1}{255}$, Eqn. 2 can be expressed as,

$$\sqrt{(\mathbf{p}-x_i')^T\Sigma_i'^{-1}(\mathbf{p}-x_i')}=\sqrt{2\ln\frac{o_i}{\alpha_\theta}}. \tag{4}$$

Therefore, we execute fine-grained online filtering by including opacity into consideration,

$$(\mathbf{p}-x_i')^T\Sigma_i'^{-1}(\mathbf{p}-x_i')=\min(2\ln\frac{o_i}{\alpha_\theta},9). \tag{5}$$

This further eliminates false-positive intersected Gaussians when performing Gaussian-tile intersections.

Scene Prefetching. Our hybrid preprocessing not only enhances rendering performance but also reduces peak GPU memory usage. However, loading Gaussian clusters at runtime could introduce non-trivial execution overhead and potentially cause frame stuttering. To address this, we prefetch future clusters ahead of time by asynchronously loading them using a dedicated GPU stream. We adopt similar prediction methods as prior works [13, 23] and find that linearly extrapolating the camera pose is an effective method for predicting future poses in our scene prefetching. This way, our approach enables asynchronous prefetching to be overlapped with runtime frame rendering. Compared to loading all Gaussian clusters into GPU memory, our scene prefetching reduces the runtime model size by 39.1% with a minimal runtime overhead (<6% of the total latency).

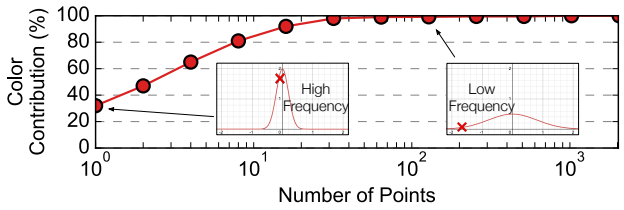


Fig. 4. The significance of Gaussians towards the final pixel. Gaussians are sorted in descending order. We empirically find that significant Gaussians are typically sampled in high-frequency regions, while insignificant Gaussians are more likely to be sampled in low-frequency regions (red crosses).

Algorithm 1: Contribution-Aware Rasterization

Data: a list of Gaussians G, a group of pixels P
Result: an image tile I

```
I.init();
for g ∈ G do
    α_0 ← ComputeTransparency(P[0], g);
    if α_0 < α_θ then
        continue;
    end
    for p ∈ P do
        α_{p,g} ← ComputeTransparency(p, g);
        I ← AccumulateColor(α_{p,g}, p);
    end
    CheckTermination();
end
```

3.4. Contribution-Aware Rasterization, CR

Motivation. The goal of our contribution-aware rasterization is to pay "computational" attention to significant Gaussians that contribute more to the final pixel values while reducing computations for less important Gaussians. Our observation is that the color values of most pixels are determined by a small fraction of Gaussians, as illustrated in Fig. 4. We characterize the MipNeRF360 dataset [4] and draw the cumulative transparency of each pixel by sorting the Gaussian transparency contribution $\Gamma\alpha$ in descending order. The result in Fig. 4 shows that 1.5% of the top Gaussians contribute to 99% of the final pixel, while remaining Gaussians contribute minimally due to the lower sampling α. Furthermore, these low contribution samples typically occur in Gaussians with larger variances or at locations farther from the mean, corresponding to the low-frequency components and sharing low gradients and good spatial locality. More rigorous analysis of the correlation between Gaussians and frequency is shown in the supplementary.

Algorithm. To leverage the observation, we propose a *contribution-aware rasterization* algorithm to identify insignificant samples and rearrange computational resources, as outlined in Algo. 1. Instead of uniformly assigning the same computation to all Gaussians, our algorithm organizes

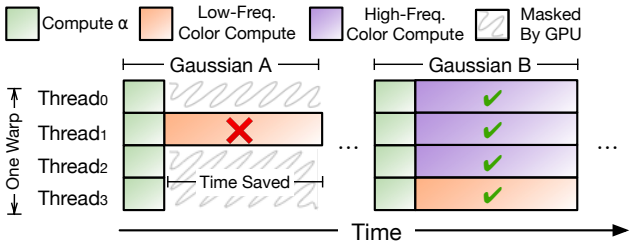


Fig. 5. An example of warp divergence in GPU. All threads compute α and then perform color blending in "lockstep". Our algorithm can detect the insignificant Gaussians (e.g., Gaussian A) and skip their color blending, as highlighted by the red cross.

every $h \times w$ pixels into a small group, P. During each iteration, only the leader pixel p within a group, which is the first pixel of this group, computes the α value for a given Gaussian, g. If this α value falls below a predefined threshold α_θ ($\frac{1}{255}$, to avoid numerical instability), the algorithm skips the color blending of this Gaussian on all pixels within P. We set both h and w to 2 to balance rendering efficiency and quality, and analyze the upper bound on the blending error introduced by this skipping in the appendix.

For Gaussians deemed important (i.e., those with α values above the threshold, α_θ), our algorithm performs the same rasterization computations as the original approach, ensuring rendering quality is preserved. This dynamic computation reassignment improves rendering efficiency without compromising visual quality.

GPU Parallelism. Not only does our algorithm reduce the overall computation, but it also inherently achieves higher parallelism by alleviating warp divergence. In the GPU implementation, pixels within a group are assigned to one warp. All pixels within a warp execute in "lockstep" as shown in Fig. 5. For instance, when blending Gaussian A, only thread 1 accumulates the insignificant Gaussian A into its pixel, while other threads wait for thread 1 to complete.

In contrast, our algorithm only lets thread 0 detect if Gaussian A is insignificant. In this case, all threads in this warp skip the color blending of Gaussian A. In the case of Gaussian B, if it is considered significant, all threads would blend Gaussian B. This ensures that these pixels either skip or process the same Gaussians collectively. This way, we eliminate divergence within a warp.

3.5. Integrated Fine-Tuning

Note that, both HP and CR modify the rendering pipeline and might impact the view consistency. To maintain a better view consistency across frames, our algorithm only requires minimal fine-tuning to achieve better rendering quality. Since both *hybrid preprocessing* and *contribution-aware rasterization* do not participate in gradient descent, these two techniques can be integrated into the current training process without any modifications to back-propagation.

Once the 3DGS model is trained by the canonical training process, we convert the model into our view-dependent scene representation in Sec. 3.3. Next, we fine-tune the model for additional 1000 steps with our loss function,

$$\mathcal{L}_{total} = \mathcal{L}_{\text{3DGS}} + \gamma * \mathcal{L}_{\text{consistency}}, \tag{6}$$

where $\mathcal{L}_{\text{3DGS}}$ is the original loss function in 3DGS and $\mathcal{L}_{\text{consistency}}$ is the view consistency loss. Here, we use ꟻLIP score [3] with 7 frames as our metric. γ is set to be 0.1.

4. Evaluation

4.1. Experimental Setup

Code Optimization. In addition to the two proposed techniques in Sec. 3, we also implement the following optimizations. Specifically, we apply fine-grained data fetching when loading Gaussian attributes to better utilize shared memory on GPU and leverage thread-level parallelism to overlap the execution of fetching Gaussian attributes and computation of alpha blending. Note that we do not claim them as our major contributions.

Datasets. To show the efficiency and robustness of SEELE, we evaluate on three datasets: Mip-NeRF360 [4], Tank&Temple [28], and DeepBlending [20]. For quality evaluation, we use PSNR, SSIM, and LPIPS. We also report FPS, the number of executed instructions (#Inst.), and the runtime model size as our performance metrics.

Baselines. To show the general applicability of our optimization technique, we apply our framework to four state-of-the-art 3DGS algorithms: 3DGS [25], MiniSplatting [11], LightGaussian [9], and AdR-Gaussian [49].

Algorithm Configurations. Unless specified otherwise, we set the number of clusters to be 24 and fix the nearest neighbor cluster, M, to be 3 in Sec. 3.3. The pixel group, P, is set to 2×2 in Sec. 3.4.

Hardware. The primary hardware we used is a mobile Ampere GPU on Nvidia AGX Orin SoC, which is the flagship development board in AR/VR and autonomous driving. Meanwhile, we also report the performance numbers on Nvidia Orin NX, a low-power embedded computing board, and Nvidia A6000, a powerful workstation GPU.

4.2. Performance and Quality

Quality Evaluation. Tbl. 1 shows the overall evaluation of SEELE on four widely-used 3DGS pipelines: 3DGS [25], MiniSplatting [11], LightGaussian [9], and AdR-Gaussian [49]. Generally, SEELE achieves better rendering quality on all three datasets across all three evaluation metrics. For instance, on average, SEELE improves PSNR and SSIM by 0.28 dB and 0.004, respectively. More qualitative comparisons are shown in the supplementary.

We also measure the view consistency metrics, ꟻLIP_1 and ꟻLIP_7 [3] in Tbl. 2. With the loss function proposed

Table 1. Quantitative evaluation of our method against the state-of-the-arts [9, 10, 25, 49]. The **bold green** results highlight the better results between ours, SEELE, and the corresponding baselines. SEELE achieves better quality across all three quality metrics with an average 2.4× speedup. 🥇 and 🥈 denote the *best* and *second-best* results among all methods, respectively.

Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Metrics	Quality			Efficiency			Quality			Efficiency			Quality			Efficiency		
	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst. (10^6)↓	Mem. (MB)↓	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst. (10^6)↓	Mem. (MB)↓	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst. (10^6)↓	Mem. (MB)↓
3DGS [25]	27.46	0.812	0.223	20.79	2168.68	710.6	23.75	0.848	0.176	41.97	1034.37	430.9	29.59	0.902	0.244	23.94	1645.11	639.9
SEELE + 3DGS	🥇 **27.72**	🥈 **0.814**	🥈 **0.217**	**59.67**	**778.25**	**380.9**	🥇 **24.02**	🥇 **0.852**	🥇 **0.168**	**127.80**	**356.15**	**207.6**	**29.79**	**0.903**	🥇 **0.240**	**86.58**	**421.74**	**400.6**
MiniSplatting [11]	27.23	0.814	0.222	71.31	797.96	145.7	23.18	0.837	0.187	143.27	329.69	83.3	🥇 **30.04**	🥇 **0.908**	0.243	120.01	382.86	154.9
SEELE + MiniSplatting	🥈 **27.70**	🥇 **0.822**	🥇 **0.212**	🥇 **131.62**	🥈 **436.41**	**106.5**	**23.74**	**0.846**	🥈 **0.179**	🥇 **268.03**	🥈 **172.40**	**60.1**	🥈 30.02	🥈 **0.908**	🥈 **0.242**	🥇 **200.62**	🥈 **224.58**	**129.6**
LightGaussian [9]	27.44	0.807	0.235	30.89	1533.50	🥈 59.4	23.82	0.842	0.189	65.60	699.71	🥈 33.8	**29.74**	**0.901**	0.250	43.17	1097.82	🥈 52.4
SEELE + LightGaussian	**27.56**	**0.810**	**0.229**	**76.36**	**589.85**	🥇 **39.1**	🥈 **23.91**	**0.846**	**0.181**	**183.86**	**259.78**	🥇 **20.7**	29.73	0.900	**0.249**	**131.16**	**324.25**	🥇 **41.4**
AdR-Gaussian [49]	26.92	0.786	0.267	65.95	885.27	288.6	23.42	0.835	0.202	114.18	442.99	203.0	**29.68**	**0.901**	0.255	78.87	485.00	330.5
SEELE + AdR-Gaussian	**27.19**	**0.790**	**0.261**	🥈 **96.25**	🥇 **327.26**	**147.6**	**23.78**	**0.838**	**0.196**	🥈 **193.83**	🥇 **154.87**	**106.2**	29.64	0.899	**0.254**	🥈 **141.87**	🥇 **178.16**	**127.7**

Table 2. The view consistency metrics on Mip-NeRF360.

	3DGS		MiniSplatting		LightGaussian		AdR-Gaussian	
	$\exists$LIP$_1$↓	$\exists$LIP$_7$↓	$\exists$LIP$_1$↓	$\exists$LIP$_7$↓	$\exists$LIP$_1$↓	$\exists$LIP$_7$↓	$\exists$LIP$_1$↓	$\exists$LIP$_7$↓
Baseline	0.0164	0.0466	0.0160	0.0460	0.0168	0.0472	0.0106	0.0332
SEELE	**0.0126**	**0.0292**	**0.0110**	**0.0300**	**0.0111**	**0.0295**	**0.0061**	**0.0169**

Table 3. The performance (FPS) over other GPUs: a low-power GPU on Nvidia Orin NX [2] and Nvidia A6000 [1].

Dataset	Mip-NeRF360		Tanks&Temples		Deep Blending	
GPU	Orin-NX	A6000	Orin-NX	A6000	Orin-NX	A6000
3DGS	3.06	134.45	6.03	197.55	3.64	155.87
SEELE + 3DGS	**9.42**	**328.98**	**19.63**	**536.77**	**14.82**	**515.60**
MiniSplatting	9.62	414.02	20.57	440.34	16.84	701.51
SEELE + MiniSplatting	**17.48**	**640.16**	**41.78**	**1108.65**	**32.40**	**1031.46**
LightGaussian	4.39	185.95	9.02	323.47	5.78	234.36
SEELE + LightGaussian	**12.72**	**431.99**	**27.45**	**755.29**	**20.38**	**656.60**
AdR-Gaussian	10.30	383.35	16.65	456.12	13.43	478.45
SEELE + AdR-Gaussian	**16.70**	**524.88**	**31.84**	**762.71**	**19.57**	**762.45**

in Sec. 3.5, SEELE achieves better view consistency against the corresponding baselines.

Performance Evaluation. With comparable rendering quality, SEELE consistently achieves speedups across all algorithms. On average, SEELE achieves 3.2×, 1.8×, 2.7×, and 1.7× speedup on 3DGS, MiniSplatting, LightGaussian, and AdR-Gaussian respectively. The higher speedup observed in 3DGS compared to the other algorithms can be attributed to the denser nature of 3DGS. This density allows SEELE to better separate irrelevant Gaussians contributed to different views using *hybrid preprocessing* introduced in Sec. 3.3, resulting in greater speedup. Despite that, sparse models like MiniSplatting still benefit from *hybrid preprocessing* due to the view-dependent redundancy, i.e., different views require different Gaussians to render. Tbl. 1 also shows the average number of executed instructions of each algorithm using NVIDIA Nsight Compute, SEELE indeed reduces the executed instructions, thus overall computation. Tbl. 4 further dissects the contributions of our techniques.

Tbl. 3 shows the speedup of SEELE on additional GPUs, Nvidia Orin NX and Nvidia A6000. The results show that the optimizations proposed in SEELE are not tied to a specific GPU architecture, exhibiting good generality across different hardware platforms. However, we observe that our optimization achieves a higher speedup on lower-power GPUs, as shown in Tbl. 3. Primarily, low-end GPUs often have more restricted hardware resources.

SEELE not only speeds up the overall rendering process but also reduces overall GPU memory consumption. Here, we exclusively focus on the GPU memory contributed by the Gaussian points, i.e., runtime model weights. Overall, results show that SEELE achieves 39.1% reduction in runtime model size across three different datasets. Even for sparse models like MiniSplatting, SEELE still saves 23.2% of model size compared to the baselines.

4.3. Ablation Study

In this section, we show the contributions of individual optimizations. Specifically, we evaluate four variants:

- +Opti.: this variant only includes the additional code optimizations proposed in Sec. 4.1.
- +Opti.+HP: this variant includes both the code optimizations in Sec. 4.1 and *hybrid preprocessing* in Sec. 3.3.
- +Opti.+CR: this variant includes code optimizations in Sec. 4.1 and *contribution-aware rasterization* in Sec. 3.4.
- SEELE: this variant is the full-fledged algorithm.

Tbl. 4 presents the results of our ablation study on 3DGS. Across all datasets, our code optimization (+Opti.) achieves a 1.1× speedup. Building on this, +Opti.+HP and +Opti.+CR yield additional 2.8× and 1.3× speedups, respectively. When all optimizations are combined, SEELE achieves an overall 3.2× speedup. All model weight savings come from HP, as the other two optimizations do not contribute to the model reduction. Note that, SEELE may increase the overall instruction count but improves the overall speedup due to the reductions on warp divergence.

In terms of the rendering quality, the additional optimizations proposed in Sec. 4.1 do not change the accuracy. The HP technique from Sec. 3.3, combined with fine-tuning as described in Sec. 3.5, further enhances rendering quality across all three quality metrics. On average, HP improves the rendering quality by 0.23 on PSNR. The effect of CR from Sec. 3.4 on quality is minimal, 0.03 in PSNR.

Table 4. The ablation study of 3DGS dissects our contributions. +Opti. refers to the code optimization in Sec. 4.1, +HP represents the *hybrid preprocessing* in Sec. 3.3, +CR represents the *contribution-aware rasterization* in Sec. 3.4.

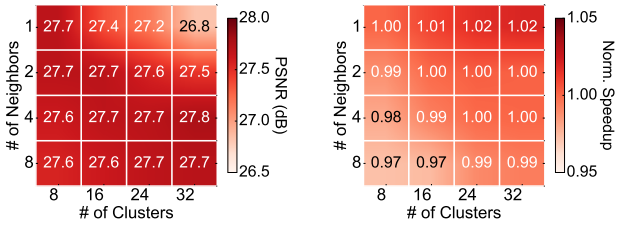
Dataset	Mip-NeRF360						Tanks&Temples						Deep Blending					
Metrics	Quality			Efficiency			Quality			Efficiency			Quality			Efficiency		
	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst.(10^6)↓	Mem.(MB)↓	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst.(10^6)↓	Mem.(MB)↓	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst.(10^6)↓	Mem.(MB)↓
3DGS [25]	27.46	0.812	0.223	20.79	2168.68	710.6	23.75	0.848	0.176	41.97	1034.37	430.9	29.59	0.902	0.244	23.94	1645.11	639.9
3DGS+Opti.	27.46	0.812	0.223	21.75	1945.88	710.6	23.75	0.847	0.176	45.96	923.51	430.9	29.59	0.902	0.244	29.60	1485.44	639.9
3DGS+Opti.+HP	27.70	0.814	0.219	46.15	814.74	380.9	24.05	0.852	0.171	118.61	374.63	207.6	29.74	0.903	0.242	81.73	**412.50**	400.6
3DGS+Opti.+CR	27.50	0.812	0.221	30.10	1574.92	710.6	23.70	0.848	0.173	57.03	763.83	430.9	29.70	0.902	0.242	38.79	1156.46	639.9
SEELE + 3DGS	**27.72**	**0.814**	**0.217**	**59.67**	**778.25**	**380.9**	**24.02**	**0.852**	**0.168**	**127.80**	**356.15**	**207.6**	**29.79**	**0.903**	**0.240**	**86.58**	421.74	**400.6**

Table 5. Quality and performance comparison on 3DGS with extra training and our finetuning. Metrics are evaluated with SEELE.

Dataset	Mip-NeRF360					
Metric	PSNR↑	SSIM↑	LPIPS↓	FPS↑	#Inst.(10^6)↓	Mem. (MB) ↓
w/ extra training	27.48	0.812	0.221	59.72	768.76	380.9
w/ fine-tuning	27.72	0.814	0.217	59.67	778.25	380.9

Table 6. Ablation study on $\mathcal{L}_{\text{consistency}}$. Lower is better. Here, we show the view consistency metric, ꟻLIP_7.

Dataset	Mip-NeRF360		Tanks&Temples		Deep Blending	
	w/ Loss	w/o Loss	w/ Loss	w/o Loss	w/ Loss	w/o Loss
3DGS	**0.058**	0.060	**0.067**	0.074	**0.042**	0.043



(a) PSNR. (b) Normalized speedup.

Fig. 6. Sensitivity of rendering quality and performance to the number of clusters and cluster neighbors in Sec. 3.3.

Fine-Tuning. Next, we present the results with and without the integrated fine-tuning, as described in Sec. 3.5. Here, we focus on the results of the Mip-NeRF360 dataset with 3DGS. Similar trends are observed across other datasets in the supplementary. Tbl. 5 shows that fine-tuning results in significant improvements. Meanwhile, fine-tuning does not affect performance for both FPS and GPU memory consumption. Tbl. 6 also shows the effectiveness of $\mathcal{L}_{\text{consistency}}$, finetuning with our proposed loss term can improve the rendering quality.

4.4. Sensitivity Study

Number of Clusters. Fig. 6 shows how rendering quality and performance vary with two key hyperparameters, the number of clusters and the number of neighboring clusters during rendering (M), in Sec. 3.3. Performance numbers are normalized to our default setting of 24 clusters with 4 neighbors (including itself). As the number of neighboring clusters increases, speedup decreases, while accuracy initially improves. However, too many clusters, e.g., 8 clusters with 8 neighbors, degrades accuracy. This degradation is because each Gaussian is responsible for too many view directions, negatively impacting rendering quality.

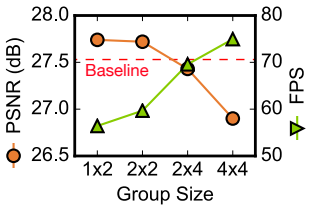


Fig. 7. Sensitivity of rendering quality and performance to the pixel group size.

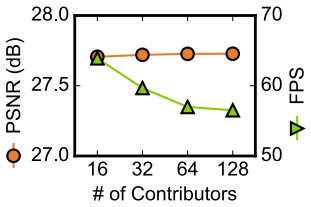


Fig. 8. Sensitivity of rendering quality and performance to the number of contributors.

Pixel Group. Fig. 7 illustrates the sensitivity of both the rendering quality and performance to the pixel group size using Mip-NeRF360 [4], as defined in Sec. 3.4. Other datasets show a similar trend. As the pixel group size increases, rendering quality initially decreases slightly, but the drop becomes more significant once the group size exceeds 2×4. By trading off between quality and performance, we ultimately select the pixel group size of 2×2.

Number of Contributors. Fig. 8 shows the sensitivity of the rendering quality and performance to the number of primary contributors in the clustering strategy (Sec. 3.3). The rendering quality is not significantly affected by the number of contributors, while the performance gradually decreases as the number of contributors increases.

5. Conclusion

Recent neural rendering has transformed the landscape of real-time rendering. 3DGS emerges as a new rendering primitive that offers unprecedented realism using the conventional rasterization pipeline. This paper proposes two principled and GPU-oriented optimizations that drastically improve the general Gaussian splatting pipelines by understanding and leveraging the GPUs' characteristics. We believe that this work could be the first step to understanding the fundamental limitations of GPUs in supporting 3DGS. The result would be valuable to rethink the next architectures tailored to this future technique.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (NSFC) Grants (62532006 and 62402312), Shanghai Qi Zhi Institute Innovation Program (SQZ202316), and Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM113).

References

[1] Nvidia rtx a6000. 7

[2] Nvidia jetson orin nx 16 gb. 7

[3] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D Fairchild. Flip: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2):15–1, 2020. 6

[4] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 1, 5, 6, 8

[5] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac: Hash-grid assisted context for 3d gaussian splatting compression. In *European Conference on Computer Vision*, pages 422–438. Springer, 2024. 2

[6] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac++: Towards 100x compression of 3d gaussian splatting. *arXiv preprint arXiv:2501.12255*, 2025. 2

[7] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16569–16578, 2023. 1

[8] Sankeerth Durvasula, Sharanshangar Muhunthan, Zain Moustafa, Richard Chen, Ruofan Liang, Yushi Guan, Nilesh Ahuja, Nilesh Jain, Selvakumar Panneer, and Nandita Vijaykumar. Contrags: Codebook-condensed and trainable gaussian splatting for fast, memory-efficient reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 28935–28945, 2025. 2

[9] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023. 2, 6, 7

[10] Guangchi Fang and Bing Wang. Mini-splatting: Representing scenes with a constrained number of gaussians. *arXiv preprint arXiv:2403.14166*, 2024. 2, 7

[11] Guangchi Fang and Bing Wang. Mini-splatting2: Building 360 scenes within minutes via aggressive gaussian densification. *arXiv preprint arXiv:2411.12788*, 2024. 2, 6, 7

[12] 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*, pages 26652–26662, 2025. 3

[13] Yu Feng, Zihan Liu, Jingwen Leng, Minyi Guo, and Yuhao Zhu. Cicero: Addressing algorithmic and architectural bottlenecks in neural rendering by radiance warping and memory optimizations. *arXiv preprint arXiv:2404.11852*, 2024. 5

[14] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *European Conference on Computer Vision*, pages 54–71. Springer, 2025. 2

[15] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996. 3

[16] Hao Gui, Lin Hu, Rui Chen, Mingxiao Huang, Yuxin Yin, Jin Yang, and Yong Wu. Balanced 3dgs: Gaussian-wise parallelism rendering with fine-grained tiling. *arXiv preprint arXiv:2412.17378*, 2024. 3

[17] Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. Ges: Generalized exponential splatting for efficient radiance field rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19812–19822, 2024. 2

[18] Alex Hanson, Allen Tu, Geng Lin, Vasu Singla, Matthias Zwicker, and Tom Goldstein. Speedy-splat: Fast 3d gaussian splatting with sparse pixels and sparse primitives. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 21537–21546, 2025. 3, 5

[19] Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 5949–5958, 2025. 2

[20] 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)*, 37(6):1–15, 2018. 1, 6

[21] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 1

[22] Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Deliege, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 3d convex splatting: Radiance field rendering with 3d smooth convexes. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 21360–21369, 2025. 2

[23] Xueshi Hou and Sujit Dey. Motion prediction and prerendering at the edge to enable ultra-low latency mobile 6dof experiences. *IEEE Open Journal of the Communications Society*, 1:1674–1690, 2020. 5

[24] Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. Efficientnerf efficient neural radiance fields. In *Proceed-*

ings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12902–12911, 2022. 1

[25] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4):1–14, 2023. 1, 6, 7, 8

[26] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics (TOG)*, 43(4):1–15, 2024. 1, 2

[27] James T Klosowski, Martin Held, Joseph SB Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998. 3

[28] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 1, 6

[29] Junseo Lee, Seokwon Lee, Jungi Lee, Junyong Park, and Jaewoong Sim. Gscore: Efficient radiance field rendering via architectural support for 3d gaussian splatting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 497–511, 2024. 3

[30] Junseo Lee, Jaisung Kim, Junyong Park, and Jaewoong Sim. Vr-pipe: Streamlining hardware graphics pipeline for volume rendering. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 217–230. IEEE, 2025. 3

[31] 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*, pages 21719–21728, 2024. 2

[32] Haolin Li, Jinyang Liu, Mario Sznaier, and Octavia Camps. 3d-hgs: 3d half-gaussian splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 10996–11005, 2025. 2

[33] Zimu Liao, Jifeng Ding, Siwei Cui, Ruixuan Gong, Boni Hu, Yi Wang, Hengjie Li, XIngcheng Zhang, Hui Wang, and Rong Fu. Tc-gs: A faster gaussian splatting module utilizing tensor cores. *arXiv preprint arXiv:2505.24796*, 2025. 3

[34] Weikai Lin, Yu Feng, and Yuhao Zhu. Rtgs: Enabling real-time gaussian splatting on mobile devices using efficiency-guided pruning and foveated rendering. *arXiv preprint arXiv:2407.00435*, 2024. 2

[35] Weikai Lin, Yu Feng, and Yuhao Zhu. Metasapiens: Real-time neural rendering with efficiency-aware pruning and accelerated foveated rendering. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 669–682, 2025. 3

[36] Yang Liu, Chuanchen Luo, Zhongkai Mao, Junran Peng, and Zhaoxiang Zhang. Citygaussianv2: Efficient and geometrically accurate reconstruction for large-scale scenes. *arXiv preprint arXiv:2411.00771*, 2024. 1

[37] Yang Liu, Chuanchen Luo, Lue Fan, Naiyan Wang, Junran Peng, and Zhaoxiang Zhang. Citygaussian: Real-time high-quality large-scale scene rendering with gaussians. In *European Conference on Computer Vision*, pages 265–282. Springer, 2025. 1

[38] Yifei Liu, Zhihang Zhong, Yifan Zhan, Sheng Xu, and Xiao Sun. Maskgaussian: Adaptive 3d gaussian representation from probabilistic masks. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 681–690, 2025. 2

[39] 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*, pages 1–11, 2024. 2

[40] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10349–10358, 2024. 2

[41] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. *arXiv preprint arXiv:2403.13806*, 2024.

[42] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):1–17, 2024. 2

[43] Minnan Pei, Gang Li, Junwen Si, Zeyu Zhu, Zitao Mo, Peisong Wang, Zhuoran Song, Xiaoyao Liang, and Jian Cheng. Gcc: A 3dgs inference architecture with gaussian-wise and cross-stage conditional processing. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*, pages 1824–1837, 2025. 3, 5

[44] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *arXiv preprint arXiv:2403.17898*, 2024. 2

[45] Sara Rojas, Jesus Zarzar, Juan C Pérez, Artsiom Sanakoyeu, Ali Thabet, Albert Pumarola, and Bernard Ghanem. Rerend: Real-time rendering of nerfs across devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3632–3641, 2023. 1

[46] Ashkan Taghipour, Vahid Naghshin, Benjamin Southwell, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Svr-gs: Spatially variant regularization for probabilistic masks in 3d gaussian splatting. *arXiv preprint arXiv:2509.11116*, 2025. 2

[47] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022. 1

[48] Jialin Wang, Rongkai Shi, Wenxuan Zheng, Weijie Xie, Dominic Kao, and Hai-Ning Liang. Effect of frame rate on user experience, performance, and simulator sickness in virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2478–2488, 2023. 1

[49] Xinzhe Wang, Ran Yi, and Lizhuang Ma. Adr-gaussian: Accelerating gaussian splatting with adaptive radius. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–10, 2024. 3, 6, 7

[50] Keyang Ye, Tianjia Shao, and Kun Zhou. When gaussian meets surfel: Ultra-fast high-fidelity radiance field rendering. *ACM Transactions on Graphics (TOG)*, 44(4):1–15, 2025. 2

[51] 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*, 37:122434–122457, 2024. 2

[52] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa volume splatting. In *Proceedings Visualization, 2001. VIS'01.*, pages 29–538. IEEE, 2001. 3