PS-GS: Group-wise Parallel Rendering with Stage-wise Complexity Reductions for Real-time 3D Gaussian Splatting

Joongho Jo
Korea University
Seoul, Republic of Korea
jojoss1004@korea.ac.kr

Jongsun Park
Korea University
Seoul, Republic of Korea
jongsun@korea.ac.kr

***Abstract*—3D Gaussian Splatting (3D-GS) is an emerging rendering technique that surpasses the neural radiance field (NeRF) in both rendering speed and image quality. Despite its advantages, running 3D-GS on mobile or edge devices in real-time remains challenging due to large computational complexity. In this paper, we introduce PS-GS, a specialized low-complexity hardware designed to enhance the pipeline parallelism of 3D-GS rendering pipeline process. In this work, we first observe that 3D-GS rendering can be parallelized when the approximate order of Gaussians, from those closest to the camera to those farthest, is known ahead. But, to enhance 3D-GS rendering speed via parallel processing, an efficient viewpoint-adaptive grouping method with low computational costs is essential. Two key computational bottlenecks of viewpoint-adaptive grouping are the grouping of invisible Gaussians and depth-based sorting. For efficient group-wise parallel rendering with low complexity viewpoint-adaptive grouping, we propose three key techniques—cluster-based preprocessing, sorting, and grouping—all seamlessly incorporated into the PS-GS architecture. Our experimental results demonstrate that PS-GS delivers an average speedup of 1.20× with negligible peak signal-to-noise ratio (PSNR) degradation.**

Keywords—Gaussian Splatting, Rendering, Accelerator

I. Introduction

3D Gaussian Splatting (3D-GS) [6] has recently emerged as a promising alternative to neural radiance fields (NeRF) [1], [2], [3], [4], [5], as it synthesizes impressive quality images while also enabling faster rendering than NeRF at high resolutions. Unlike NeRF, 3D-GS explicitly models scenes using millions of Gaussians with learnable attributes, and it rasterizes them for rendering. 3D-GS avoids time-consuming inference of sampling points and it allows for parallel processing, making rendering faster and more efficient. However, while 3D-GS based real-time rendering has been demonstrated on powerful server-class GPUs due to low computation cost, its application to edge devices, such as AR and VR headsets, still remains challenging. For example, on NVIDIA's Orin Nano SoC with a mobile Ampere GPU, 3D-GS struggles to render at 5 frames per second (FPS) for an 800×800 frame, which is insufficient for real-time interactions.

3D-GS rendering generates images by processing millions of Gaussians through three stages of operations: preprocessing, sorting, and rasterization stages. Many previous works on 3D-GS rendering have primarily focused on improving speed by intuitively reducing the number of Gaussians or by employing compression techniques, such as quantization. In [7], the Gaussians that do not significantly contribute to the synthesized image quality are removed, and they also employ compression techniques like adaptive quantization. Similarly, a volume-based masking strategy is utilized in [9], and [10] offers a color-cued densification and learnable pruning mask to further reduce the number of Gaussians. Sensitivity-based vector grouping and entropy encoding are introduced in [8] to compress Gaussian parameters. Although those approaches intuitively reduce the overall complexity of the 3D-GS rendering process, algorithmic approaches to enhance pipeline parallelism or to optimize the efficiency of the rendering pipeline itself had never been explored.

Recently, GSCore [11] employs hierarchical sorting to enhance pipeline parallelism in the sorting stage. It also introduces Gaussian shape-aware intersection tests and skipping ineffective computation technique to reduce computational complexity in the rasterization stage. However, the pipeline parallelism of the overall 3D-GS rendering process has not been considered, and only the sorting stage has been addressed. Moreover, the improvement in pipeline parallelism during the sorting stage is limited by the additional computations introduced through approximation in the hierarchical sorting. Furthermore, GSCore does not address the complexity of the preprocessing stage, which has a comparable computational burden to that of the sorting stage in 3D-GS rendering. To further accelerate 3D-GS rendering process, in-depth research is highly required to improve parallelism and to further reduce computational complexity across the entire pipeline. To the best of our knowledge, this is the first work to analyze and propose techniques for pipeline parallelism across the entire 3D-GS rendering pipeline.

In this paper, we introduce PS-GS, a specialized low-complexity hardware designed to enhance the pipeline parallelism of 3D-GS rendering. Our analysis demonstrates that 3D-GS rendering can be parallelized when the approximate depth order of the Gaussians, from nearest to farthest based on the current viewpoint, is known ahead. As a low complexity viewpoint-adaptive grouping is crucial to enhance 3D-GS rendering speed through parallel processing, we employ cluster-based preprocessing, sorting, and grouping. Cluster-based preprocessing is for identifying invisible Gaussians, while cluster-based sorting is designed to perform efficient approximate sorting. Cluster-based grouping ensures efficient viewpoint-adaptive grouping by dynamically adjusting the process based on the current viewpoint. Based on the proposed algorithmic approaches, we present a novel hardware architecture called PS-GS, specialized for 3D-GS rendering. The experimental results show that PS-GS consistently achieves an average speedup of 1.20× with negligible peak signal-to-noise ratio (PSNR) loss.

II. Backgound

In this section, we first present a brief overview of novel view synthesis and existing methods, with a particular focus on radiance field techniques. Then, we introduce 3D Gaussian Splatting (3D-GS), the current state-of-the-art radiance field rendering technique.

978-3-9826741-0-0/DATE25/© 2025 EDAA

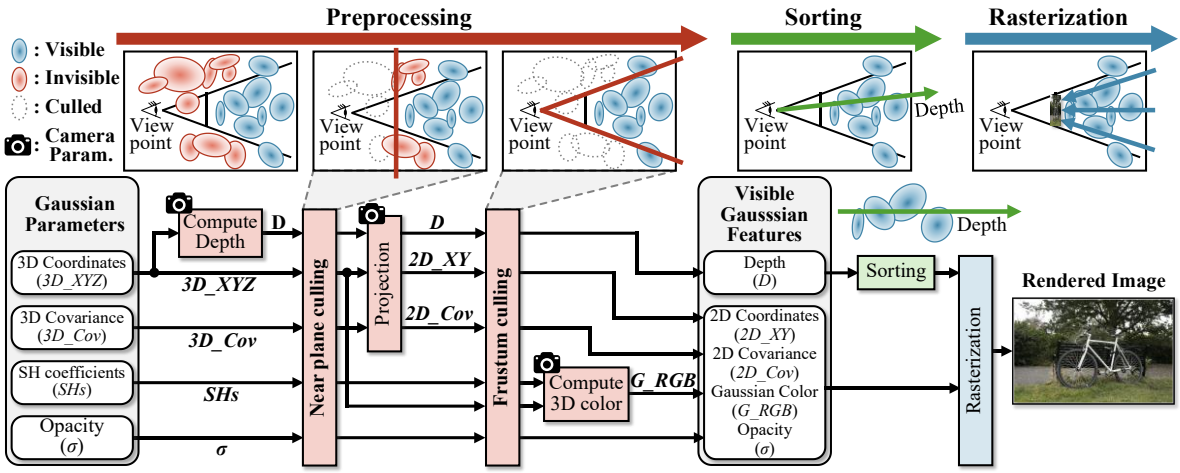

Fig. 1: The overall rendering pipeline of 3D Gaussian Splatting (3D-GS).

A. Novel View Synthesis

Novel view synthesis (NVS) plays a crucial role in 3D reconstruction by generating new images of a scene from unseen viewpoints based on input images. Conventional NVS methods used explicit representations like light fields [17], meshes [18], [19], and voxels [20], [21] to model 3D scenes. While former NVS techniques work well in certain cases, they often struggle with complex scenes and sparse inputs, leading to information loss and incomplete reconstructions. Neural Radiance Fields (NeRF) [1] has recently emerged as a promising method for NVS. NeRF implicitly represent scenes using a multi-layer perceptron (MLP), which estimate the color and density of points in 3D space along each ray. This implicit representation achieves high-quality image synthesis. One of the main difficulties encountered NeRF is huge amount of computations that makes its real-time processing almost impossible. Previous works [2], [22], [23], [24], [25] have focused on accelerating NeRF with methods such as voxel grids or hash tables. However, achieving real-time performance, even on server-class GPUs such as the NVIDIA RTX 4090 or A100, remains a significant challenge.

B. 3D Gaussian Splatting

3D Gaussian Splatting (3D-GS) [6] is an innovative technique in the field of NVS, offering both high-quality visual output while also enabling faster rendering than NeRF. 3D-GS explicitly models scenes using millions of Gaussians, and it rasterizes them for rendering. Fig. 1 shows the overall rendering pipeline of 3D-GS. 3D-GS rendering pipeline consists of preprocessing, sorting, and rasterization stages. First, the inputs of preprocessing are Gaussians, which is defined by 4 parameters: a 3D coordinates ($3D\_XYZ$) representing its center, a 3×3 3D covariance matrix ($3D\_Cov$) capturing its shape and orientation, an opacity (σ), and spherical harmonic coefficients (SHs) representing its color and appearance from different viewing angles. The parameters of each Gaussian are learned during training using gradient-based optimization techniques.

Preprocessing. The preprocessing stage identifies and culls invisible Gaussians based on the current viewpoint, while simultaneously projecting remaining 3D Gaussians onto the 2D image plane. First, depth (D) is computed from $3D\_XYZ$ using camera parameters. Based on D, near plane culling removes Gaussians that are too close and therefore invisible. Next, surviving Gaussians are projected onto the 2D image plane using D, $3D\_XYZ$ and $3D\_Cov$, resulting in 2D coordinates ($2D\_XY$) and 2D covariance ($2D\_Cov$). Frustum culling is then applied to discard Gaussians outside the camera's field of view. Finally, Gaussian color (G_RGB) is computed using the SHs of visible Gaussians. After preprocessing, the projected visible Gaussians are defined by their D, $2D\_XY$, $2D\_Cov$, G_RGB, and σ.

Sorting. To calculate pixel colors, as with conventional rendering techniques, rasterization is performed starting from the nearest objects (Gaussians in 3D-GS) to the current viewpoint. Therefore, in the sorting stage, the Gaussians are sorted based on the depth (D) values computed during preprocessing stage.

Rasterization. Rasterization stage calculates pixel colors by utilizing the features of visible Gaussians, including D, $2D\_XY$, $2D\_Cov$, G_RGB, and σ. The rasterization stage can be primarily divided into two stages: α-computation, followed by α-blending with early-exit mechanism. α represents the opacity of the Gaussian projected onto a pixel. α value for the i-th Gaussian is calculated using the following equation:

$$\alpha_i = \sigma_i * \exp\left(-\frac{1}{2}(P - 2D\_XY_i)^T {2D\_Cov_i}^{-1}(P - 2D\_XY_i)\right), \quad (1)$$

where P is the pixel's coordinates, and ${2D\_Cov_i}^{-1}$ is the inverse of i-th Gaussian's 2D covariance matrix. In α-blending step, pixel colors are computed starting from the one closest to the viewpoint, using the following equation:

$$Pixel\ Color = \sum_{i=1}^{N} G\_RGB_i \alpha_i \prod_{k=1}^{i-1}(1 - \alpha_k), \quad (2)$$

where N is the number of visible Gaussian associated with the pixel. During the accumulation operation in (2), an early-exit is triggered if $\prod_{k=1}^{i-1}(1 - \alpha_k)$ falls below a predefined threshold (e.g., 10^{-4} as suggested in [6]).

3D-GS employs a traditional tile-based rendering approach [12], to improve the rendering speed. The rendered image is partitioned into numerous 16×16 pixel tiles, and sorting and rasterization operations are executed on each tile independently.

III. Motivation

In this section, we first profile the 3D-GS rendering pipeline. We then identify and discuss our key observations regarding pipeline parallelism of 3D-GS.

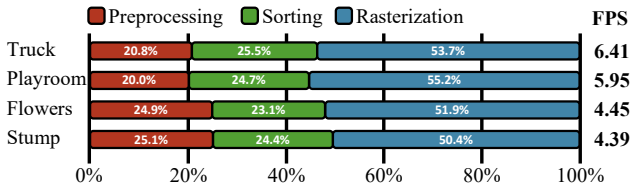

Fig. 2: The normalized runtime breakdown and FPS of 3D-GS.

A. Profiling of the 3D-GS rendering pipeline

To analyze the rendering performance of 3D-GS, we conduct a profiling study on an edge GPU, NVIDIA's Orin Nano SoC. Fig. 2 presents the normalized runtime breakdown and FPS for the primary stages of 3D-GS rendering—preprocessing, sorting, and rasterization—across the truck, playroom, flowers, and stump scenes. Each model is trained with 30k iterations. Detailed information about the dataset is provided in section VI-A. As illustrated in Fig. 2, preprocessing, sorting, and rasterization account for an average of 22.8%, 24.4%, and 52.8% of the total rendering time, respectively. The FPS results in the right demonstrate the difficulty of achieving 6 FPS, highlighting the limitations of applying 3D-GS to real-time applications, such as AR and VR, when running 3D-GS on edge GPUs. In sections III-B and III-C, we evaluate the pipeline parallelism of the rendering process and analyze the computational bottlenecks at viewpoint-adaptive grouping to identify the opportunities for accelerating 3D-GS rendering.

B. Pipeline Parallelism of 3D-GS Rendering Process

Fig. 3 illustrates the execution diagram for (a) baseline 3D-GS, (b) GSCore [11], and (c) the proposed group-wise parallel rendering. Culling step in the preprocessing stage is essential for reducing the computational complexity of the 3D-GS rendering process before sorting stage. Since rasterization starts with the Gaussians nearest to the current viewpoint, a sorting operation based on depth values is necessary prior to rasterization stage. Due to the interdependencies between the preprocessing, sorting, and rasterization stages, baseline 3D-GS, as shown in Fig. 3 (a), is executed sequentially across these three stages. Recently, to improve pipeline parallelism, the state-of-the-art hardware GSCore [11] employs hierarchical sorting to enable the overlap of sorting and rasterization stages, which is shown in Fig. 3 (b). However, the hierarchical sorting has two main drawbacks: first, GSCore does not address preprocessing parallelism, and another drawback is a high computational cost due to ap proximate sorting . A detailed analysis of GSCore's approximate sorting is provided in section VI-B.

Since the pixel color in (2) is a simple accumulation operation, if $G\_RGB_i\alpha_i\prod_{k=1}^{i-1}(1-\alpha_k)$ is replaced with F_i, and then divided into M groups $(G_0, G_1, G_2, \dots, G_M)$, it becomes as follows:

$$Pixel\ Color = \sum_{i\in G_0} F_i + \sum_{i\in G_1} F_i + \sum_{i\in G_2} F_i + \cdots + \sum_{i\in G_M} F_i. \quad (3)$$

As shown in Fig. 3 (c), group-wise parallel rendering can be implemented in the 3D-GS pipeline. However, to avoid image quality degradation, the grouping should be carefully tailored to current viewpoint. Fig. 4 highlights the potential challenges in the grouping process. On the left side of Fig. 4, the grouping is optimized for viewpoint 1, where computation proceeds in the order of Group 1, Group 2, and Group 3, based on the distance from th e viewpoint, to ensure correct results. On the right side, however, if the grouping from viewpoint 1 is

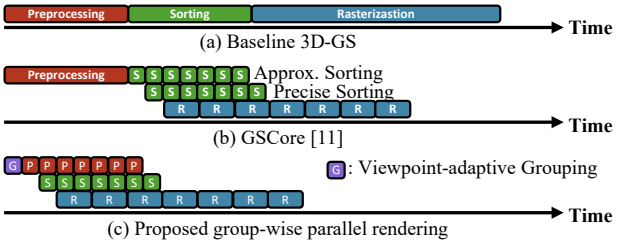

Fig. 3: Execution diagram for (a) Baseline 3D-GS, (b) GSCore [11], and (c) Proposed group-wise parallel rendering.

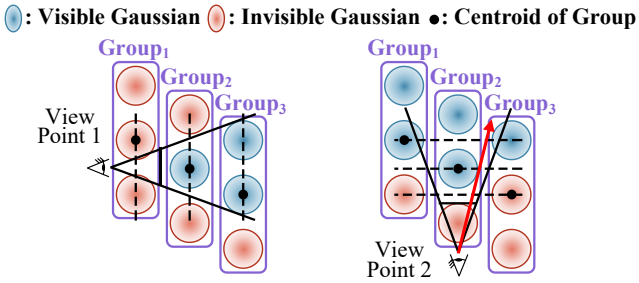

Fig. 4: The potential challenges in the grouping process.

TABLE I. **PROPORTIONS OF COMPUTATIONAL WASTES ACROSS PREPROCESSING STEPS**

	Truck	Bicycle	Flowers	Stump
Depth & Near plane culling	65.46%	75.50%	76.44%	81.09%
Projection & Frustum culling	48.93%	64.48%	65.37%	68.74%
Overall waste in preprocessing	**40.04%**	**55.39%**	**56.39%**	**60.47%**

applied to viewpoint 2, computation occurs in the order of Group 3, Group 2, and Group 1, based on their distance from viewpoint 2. In this case, the pixel marked by the red arrow should be computed in the order of Group 2 and Group 3, but the current grouping may result in incorrect color computations for that pixel. To address this issue, viewpoint-adaptive grouping is important, as shown in Fig 3 (c). *(Observation 1) To enhance 3D-GS rendering speed through parallel processing, an efficient viewpoint-adaptive grouping method with low computational costs is crucial.*

C. The Computational Bottleneck of Viewpoint-adaptive Grouping

The first computational bottleneck in viewpoint-adaptive grouping is that unnecessary computations are performed on invisible Gaussians, as the grouping is conducted before the preprocessing stage. Table I shows the percentage of the computational wastes caused by processing invisible Gaussians. Here, the computational wastes means the unnecessary computations that do not contribute to the final image during each preprocessing step. As shown in Table I, for stump scene, 81.09% of depth calculations and near-plane culling involve invisible Gaussians. Similarly, 68.47% of the projection and frustum culling operations are redundant. Overall, 60.47% of preprocessing computations are wasted due to the processing of invisible Gaussians. Viewpoint-adaptive grouping is also affected by unnecessary computations related to invisible Gaussians, as shown in Group 1 of Fig. 4. *(Observation 2) To address the first computational bottleneck, an efficient method for detecting invisible Gaussians is essential.*

As shown in Fig. 4, viewpoint-adaptive grouping should be performed based on the depth from current viewpoint. So, to do this, the computational overhead is sorting Gaussians by

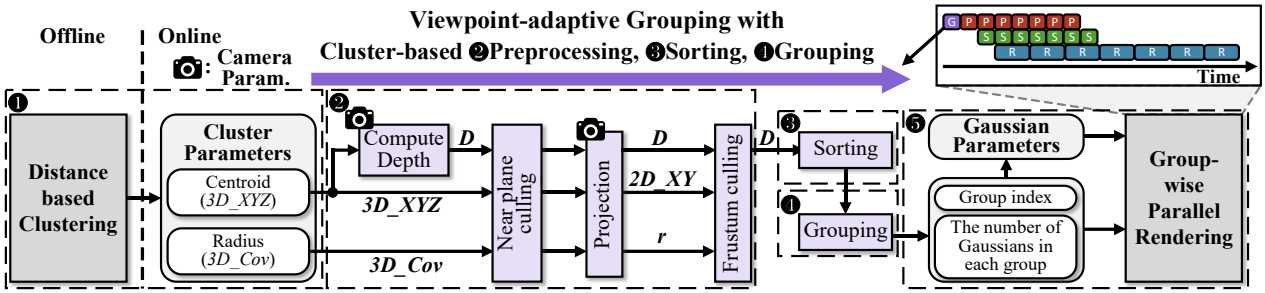

Fig. 5: The overall process of the proposed group-wise parallel rendering with cluster-based preprocessing, sorting, and grouping.

depth. In the case of stump scene, the model contains 4.9 million Gaussians, and depending on the viewpoint, up to 2.0 million visible Gaussians may require sorting, which is substantial overhead. *(Observation 3) For cost-effective viewpoint-adaptive grouping, an efficient approximate Gaussian sorting method is needed.*

IV. The Proposed Group-wise Parallel Rendering with Low Complexity Viewpoint-adaptive Grouping

In this section, we introduce the group-wise parallel rendering with viewpoint-adaptive grouping (PS-GS). Based on our three observations, to reduce the computational complexity of viewpoint-adaptive grouping, we employ the cluster-based preprocessing, sorting, and grouping. Cluster-based preprocessing is an efficient method for identifying invisible Gaussians, while cluster-based sorting is designed to perform efficient approximate sorting. Finally, cluster-based grouping enables efficient viewpoint-adaptive grouping with reduced computational cost. All three techniques share the same clusters. Fig. 5 illustrated the overall process of the proposed group-wise parallel rendering.

A. The Overall Process of PS-GS

Clustering: We employ a K-means clustering algorithm that groups Gaussians based on distance, assuming spherical clusters. The purpose of employing distance-based clustering is to group nearby Gaussians for efficient detection of those invisibles from current viewpoint. Spherical clusters are chosen to treat them as spherical Gaussians in cluster-based preprocessing. To account the size of Gaussians, the radius of cluster sphere can be calculated as following:

$$R_j = \max_{i \in j}\left(dist(centroid_j, 3D\_XYZ_i)\right) + SG_i). \quad (4)$$

In (4), R_j represents the radius of j-th cluster, and $dist(centroid_j, 3D\_XYZ_i)$ refers to the distance between the centroid of j-th cluster and the 3D position of i-th Gaussian. SG_i means the size of i-th Gaussian, determined using the 3-sigma rule. As shown in step ❶ of Fig. 5, clustering is performed offline using a pre-trained model, while in the online (real-time) phase, viewpoint-adaptive grouping is conducted using cluster parameters (centroid and radius). Since the cluster is treated as a spherical Gaussian, the 3D covariance matrix can be simply calculated using 3-sigma rule and R_j as follows:

$$3D\_Cov = \begin{bmatrix} (R_j/3)^2 & 0 & 0 \\ 0 & (R_j/3)^2 & 0 \\ 0 & 0 & (R_j/3)^2 \end{bmatrix}, \quad (5)$$

where $3D\_Cov$ represents the 3D covariance matrix of j-th cluster, while R_j denotes the radius of j-th cluster in (4).

Cluster-based preprocessing, sorting, and grouping: As clusters are treated as Gaussians, clusters can be processed similarly to the preprocessing stage in 3D-GS (step ❷ of Fig. 5). The main difference from 3D-GS preprocessing is that, for viewpoint-adaptive grouping, only culling and projection are needed, and 3D color computation is not necessary. Once cluster-based preprocessing is complete, sorting is performed based on the depth of remaining clusters (step ❸ of Fig. 5). Based on the sorted order, the clusters are grouped in the sequence according to their proximity to the viewpoint (step ❹ of Fig. 5).

Group-wise parallel rendering: Using the group indices generated through viewpoint-adaptive grouping, along with the number of Gaussians in each group, group-wise parallel rendering is performed, starting with the Gaussians in the groups closest to the viewpoint (step ❺ of Fig. 5).

V. PS-GS Hardware Architecture

A. Architecture Overview

Fig. 6 illustrates the overall hardware architecture of PS-GS and the detailed block diagrams. Fig. 6 (a) presents the overall design, consisting of five key hardware modules: preprocessing module 1 (PM1), preprocessing module 2 (PM2), sorting module (SM), rasterization module (RM), and grouping module (GM). All the modules except GM are implemented in four instances. Although the diagram is simplified, all the modules receive the number of Gaussians to be processed as parameters. The buffer is shared by PM1, PM2, SM, RM, and GM, and consists of three components: 1) Gaussian parameter buffer (G Param.): stores the parameters of Gaussians. 2) Camera parameter buffer (C Param.): stores the current viewpoint's camera parameters, including image resolution (W, H), field of view (Θx, Θy), 4×4 view matrix (Vm), and 4×4 projection matrix (Pm). 3) Gaussian feature buffer (G Feat.): stores the Gaussian features generated by all modules.

As shown in Fig. 6 (b), to efficiently support cluster-based preprocessing without incurring additional area overhead, preprocessing stage of 3D-GS is divided into PM1 and PM2. PM1 handles depth computation, projection, near plane culling, and frustum culling during 3D-GS preprocessing stage. Those operations are essential for both 3D-GS and cluster-based preprocessing, allowing PM1 to be utilized in both processes. **PM1** consists of four inner product units (IPUs), two 2D position computation units (PCUs), one 2D covariance computation unit (CCU), and a near plane culling and frustum culling unit (NPC & FC Unit). Fig. 6 (e), (f), and (g) illustrate the detailed IPU, PCU, and NPC & FC Units, respectively. IPU performs a dot product between a 3-element vector and a 4-element vector, with the fourth element of the 4-element vector used only in addition. PCU adjusts the projected coordinates to match pixel coordinates. NPC & FC units perform near plane culling using the calculated depth (D), and frustum culling is conducted based on the projected 2D

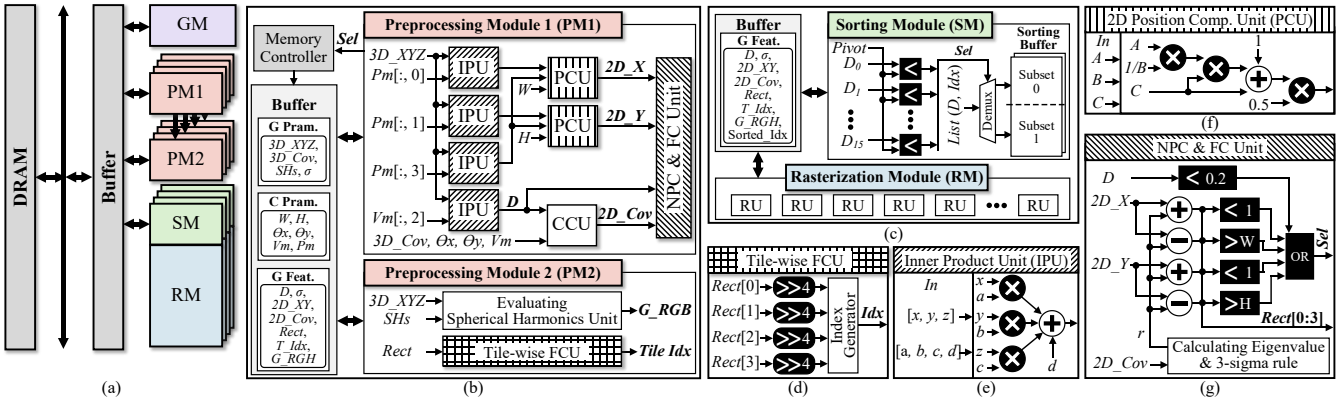

Fig. 6: The overall hardware architecture of PS-GS and the detailed block diagram of each module.

coordinates (*2D_XY*) and the radius (*r*) calculated using the eigenvalues of the projected Gaussian's covariance (*2D_Cov*) and the 3-sigma rule. The CCU is a matrix multiplication unit comprised of 18 multipliers. **PM2** features a dedicated hard ware unit with 9 multipliers to evaluate spherical harmonics polynomials of degree 3, producing Gaussian color (*G_RGB*). It also utilizes the size of the Gaussian (*Rect*), calculated by the NPC & FC Unit of PM1, in a tile-wise frustum culling unit (Tile-wise FCU) to distribute Gaussians across tiles (Fig. 6 (d)).

SM sorts the Gaussians distributed across tiles based on the depth, performing both the 3D-GS sorting stage and cluster-based sorting. It consists of 16 comparators and implements the quicksort algorithm with a randomly selected pivot value (Fig. 6 (c)). The comparators simultaneously compare the depths (*D*) with pivot. Depth values less than the pivot, along with their corresponding indices (*idx*), are stored in the subset 0 of the sorting buffer. Those greater than the pivot are stored in subset 1. This process is repeated to execute the quicksort algorithm. **RM** consists of 16 rasterization units (RUs), each composed of 12 multipliers for α-computation (1) and α-blending (2) as introduced in section II-A on rasterization. RU also includes the dedicated hardware for an exponential function using a 5th-order Taylor series approximation. Given that the tile size is 16×16 pixels (as noted in [6]), each RU processes 16 pixels. The color accumulation operation in (2) is performed using the sorted indices (*Sorted_Idx*) generated by the SM.

B. PS-GS Overall Dataflow

As mentioned in section IV-A, clusters are treated as Gaussians. So, PM1 handles cluster-based preprocessing using the centroids and covariance matrices that are generated offline for viewpoint-adaptive grouping. The depth values and cluster indices are then passed to the SM for cluster-based sorting. Finally, the sorted cluster indices, based on depth, along with the number of Gaussians within each cluster, are passed to GM for cluster-based grouping. Using the group indices, the number of Gaussians per group, and their corresponding indices, group-wise parallel rendering is performed.

VI. Experimental Results

A. Methodology

Algorithm. The performance of the proposed techniques have been evaluated using the two scenes from the Tanks&Temples dataset (T&T) [13], the two scenes from the Deep Blending (DB) [14], and seven sets of scenes presented in Mip-NeRF360 (Mip360) [15]. 3D-GS-30k pre-trained model with an RTX A6000 GPU is used for simulations. A train/test split datasets is employed, following the methodology proposed by Mip-NeRF360, where every 8th photo is used for testing. In addition, to improve the throughput and area efficiency of PS-GS, the models trained in 32bit-floating point are converted to 16bit-floating point. To generate error metrics, standard peak signal-to-noise ratio (PSNR) is used. The number of clusters is set to 0.128% of the total number of Gaussians in the model, and the number of groups is fixed at 5 for all the models. **Hardware.** The proposed PS-GS architecture is implemented in RTL and synthesized using Synopsis Design Compiler using 28nm CMOS technology. Power and energy consumption are simulated using Synopsys PrimeTime PX. The speedup is obtained using a cycle-level simulator, with a DRAM bandwidth of 51.2GB/s, and the DRAM energy is computed based on [16].

TABLE II. **Computation Reduction of Cluster-Based Rreprocessing**

Datasets		T&T	DB	Mip360
Average # of Gaussians		1.78M	2.98M	3.36M
Cluster-based Preprocessing Efficiency		70.04%	89.23%	87.43%
Comp. Reduction	w/o Parallel	30.80%	72.13%	55.11%
	w/ Parallel (Overlapping)	82.63%	92.94%	88.69%

B. Experimental Results

Table II presents the average number of Gaussians, the culling efficiency (indicating how effectively the proposed cluster-based preprocessing technique identifies invisible Gaussians), and the computational reductions in the preprocessing stage with and without group-wise parallel processing. For DB dataset, 89.23% of invisible Gaussians are identified through cluster-based preprocessing and excluded from the rendering process in advance. Without parallel rendering (w/o parallel), the computation load, including the computational overhead of cluster-based preprocessing, is reduced by 72.13% compared to 3D-GS preprocessing stage. When operating with parallel rendering (w/ parallel), overlapping further conceals the computation, resulting in a similar effect of 92.94% reduction of computational load. These results demonstrate that even without parallel rendering, the cluster-based preprocessing alone can significantly reduce the preprocessing computations. When combined with parallel rendering, it achieves an average reduction of 80.09% in preprocessing computations.

Fig. 7 (a) compares the cycle counts of GSCore's hierarchical sorting and the proposed cluster-based sorting. All the cycle counts are normalized to the number of cycles required for sorting in 3D-GS. And assuming the same

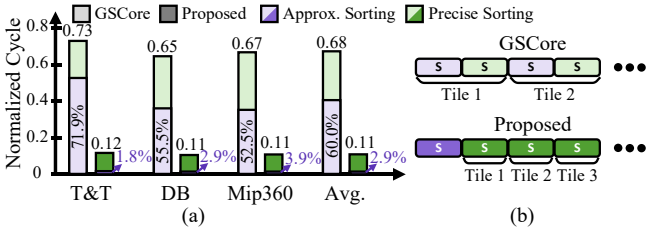

Fig. 7: (a) Normalized cycle counts of GSCore's hierarchical sorting and the proposed cluster-based sorting. (b) Execution diagram of sorting stage.

TABLE IV. **COMPARISON OF RENDERING QUALITY**

	Ori.	PS-GS	Ori.	PS-GS	Ori.	PS-GS	Ori.	PS-GS
Datasets	Tanks&Tamples				Deep Blending			
Scene	Train		Truck		Playroom		Drjohnson	
PSNR↑	21.77	21.76	24.94	24.87	29.93	29.79	28.94	28.77
Datasets	Mip-NeRF360							
Scene	Bicycle		Bonsai		Counter		Flowers	
PSNR↑	25.17	25.16	31.98	31.89	28.89	28.77	21.45	21.44
Datasets	Mip-NeRF360						**Avg.**	
Scene	Garden		Stump		Treehill			
PSNR↑	27.18	26.92	26.56	26.56	22.30	22.30	**26.28**	**26.20**

number of comparators are used for all. Fig. 7 (b) illustrates the execution diagram of the sorting stage for both GSCore and PS-GS. Across all three datasets, GSCore reduces approximately 32% of sorting cycles compared to baseline 3D-GS, thanks to the overlapping effect of hierarchical sorting. In contrast, the proposed cluster-based sorting hides approximately 89% of the cycles on average. This difference arises because, as illustrated in Fig. 7 (b), GSCore performs approximate sorting for each tile, whereas the proposed cluster-based sorting only requires a single approximate sorting, after which only precise sorting is needed for each tile's computation. For instance, in the bicycle scene of the Mip360 dataset, rendering an image with a resolution of 1237×822 requires 4056 tiles. Given this large number of tiles, performing approximate sorting for each tile leads to a substantial computational burden. As a result, approximate sorting accounts for about 60.0% of the sorting cycles in GSCore, whereas it is only 2.9% in the proposed method.

Table III presents the PSNR results for each scene rendered using 3D-GS (Ori.) and the proposed PS-GS with group-wise parallel rendering. Across the 11 scenes analyzed, it is observed that the PSNR decreases by only 0.08 on average when using PS-GS compared to the original 3D-GS.

C. *Hardware Evaluation*

Table IV shows the synthesized results of PS-GS. The area of PS-GS is $3.874mm^2$, and power is $1.039W$ with the operation frequency of 1GHz. Fig. 8 (a) compares the performance of the proposed hardware with and without group-wise parallel rendering applied, as well as with GSCore. PS-GS with parallel rendering achieves an average speedup of 1.75× and 1.2× over PS-GS without parallel rendering and GSCore, enabling real-time rendering at 117 FPS. However, as illustrated in Fig. 8 (b), GSCore focuses more on eliminating unnecessary computations rather than enhancing the parallelism of 3D-GS. As a result, while the proposed PS-GS outperforms GSCore in terms of speed, its energy efficiency is comparable or slightly lower.

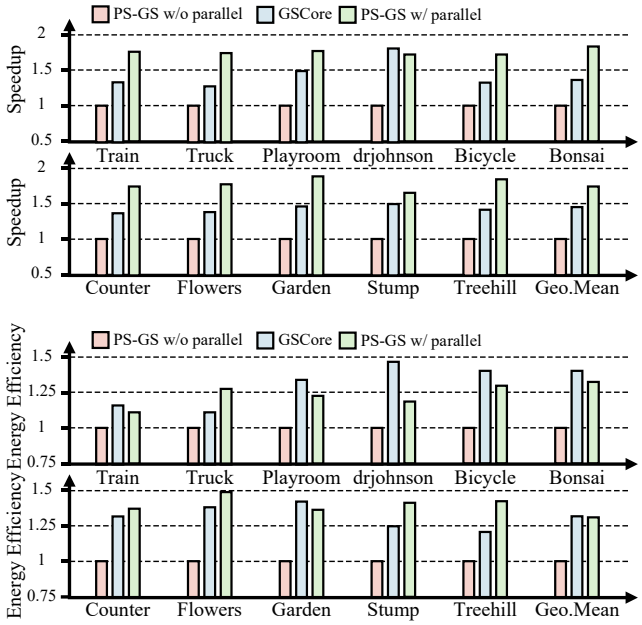

Fig. 8: (a) Speedup of PS-GS over the GSCore. (b) Normalized end-to-end energy efficiency.

TABLE III. **HARDWARE CONFIGURATION**

Module	Configuration	Area $[mm^2]$	Power $[W]$
PM1	4	0.258	0.148
PM2	4	0.487	0.280
SM & GM	4, 1	0.067	0.053
RM	4	1.742	0.325
Buffer	4×2×40KB	1.320	0.233
Total		3.874	1.039
Operating Freq.		1GHz	

VII. CONCLUSION

In this paper, we introduce PS-GS, a specialized low-complexity hardware designed to enhance the pipeline parallelism of 3D Gaussian Splatting (3D-GS) rendering through viewpoint-adaptive grouping. PS-GS leverages cluster-based preprocessing, sorting, and grouping to effectively reduce the computational overhead of viewpoint-adaptive grouping, thus enabling efficient group-wise parallel processing of 3D-GS while maintaining image quality (PSNR). Experimental results demonstrate that PS-GS outperforms state-of-the-art 3D-GS hardware across a variety of scenes.

ACKNOWLEDGMENT

This work was supported in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2024-00345481); in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2024-00405495); in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2022-0-00266, Development of Ultra-Low Power Low-Bit Precision Mixed-Mode SRAM PIM); in part by the Ministry of Trade, Industry and Energy(MOTIE) and Korea Institute for Advancement of Technology(KIAT) through the "International Cooperative R&D program"(Task No. P0028486)

REFERENCES

[1] B. Mildenhall et al, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis", In ECCV, 2020.

[2] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," ACM Trans. Graph., vol. 41, no. 4, pp. 1–15, Jul. 2022.

[3] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields," in 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada: IEEE, Oct. 2021, pp. 5835–5844.

[4] S. Li, H. Li, Y. Wang, Y. Liao, and L. Yu, "Steernerf: Accelerating nerf rendering via smooth viewpoint trajectory," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023, pp. 20701–20711.

[5] C.-Y. Lin, Q. Fu, T. Merth, K. Yang, and A. Ranjan, "Fastsr-nerf: Improving nerf efficiency on consumer devices with a simple super-resolution pipeline," in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2024, pp. 6036–6045.

[6] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering.," ACM Trans. Graph., vol. 42, no. 4, pp. 139–1, 2023.

[7] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang, "LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS," Mar. 29, 2024, arXiv.

[8] S. Niedermayr, J. Stumpfegger, and R. Westermann, "Compressed 3d gaussian splatting for accelerated novel view synthesis," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 10349–10358.

[9] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park, "Compact 3d gaussian representation for radiance field," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 21719–21728.

[10] S. Kim, K. Lee, and Y. Lee, "Color-cued Efficient Densification Method for 3D Gaussian Splatting," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 775–783.

[11] J. Lee, S. Lee, J. Lee, J. Park, and J. 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 (ASPLOS), Volume 3, La Jolla CA USA: ACM, Apr. 2024, pp. 497–511.

[12] M. Terai, J. Fujiki, R. Tsuruno, and K. Tomimatsu, "Tile-Based Modeling and Rendering," in Smart Graphics, vol. 4569.

[13] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: benchmarking large-scale scene reconstruction," ACM Trans. Graph., vol. 36, no. 4, pp. 1–13, Aug. 2017.

[14] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," ACM Trans. Graph., vol. 37, no. 6, pp. 1–15, Dec. 2018.

[15] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 5470–5479.

[16] Z. Zhou, J. Liu, Z. Gu, and G. Sun, "Energon: Toward efficient acceleration of transformers using dynamic sparse attention," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 1, pp. 136–149, 2022.

[17] M. Broxton et al., "Immersive light field video with a layered mesh representation," ACM Trans. Graph., vol. 39, no. 4, Aug. 2020.

[18] A. Collet et al., "High-quality streamable free-viewpoint video," ACM Trans. Graph., vol. 34, no. 4, pp. 1–13, Jul. 2015.

[19] Z. Su, L. Xu, Z. Zheng, T. Yu, Y. Liu, and L. Fang, "RobustFusion: Human Volumetric Capture with Data-Driven Visual Cues Using a RGBD Camera," in ECCV, 2020.

[20] K. Guo et al., "The relightables: volumetric performance capture of humans with realistic relighting," ACM Trans. Graph., vol. 38, no. 6, pp. 1–19, Dec. 2019.

[21] T. Hu, T. Yu, Z. Zheng, H. Zhang, Y. Liu, and M. Zwicker, "Hvtr: Hybrid volumetric-textural rendering for human avatars," in 2022 International Conference on 3D Vision (3DV), IEEE, 2022, pp. 197–208.

[22] Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "TensoRF: Tensorial Radiance Fields," in ECCV, 2022.

[23] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 5501–5510.

[24] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," Advances in Neural Information Processing Systems, vol. 33, pp. 15651–15663, 2020.

[25] C. Sun, M. Sun, and H.-T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 5459–5469.