Characterization and Analysis of the 3D Gaussian Splatting Rendering Pipeline

Jiwon Lee, Yunjae Lee, *Graduate Student Member, IEEE*, Youngeun Kwon, and Minsoo Rhu, *Senior Member, IEEE*

***Abstract*—Novel view synthesis, a task generating a 2D image frame from a specific viewpoint within a 3D object or scene, plays a crucial role in 3D rendering. Neural Radiance Field (NeRF) emerged as a prominent method for implementing novel view synthesis, but 3D Gaussian Splatting (3DGS) recently began to emerge as a viable alternative. Despite the tremendous interest from both academia and industry, there has been a lack of research to identify the computational bottlenecks of 3DGS, which is critical for its deployment in real-world products. In this work, we present a comprehensive end-to-end characterization of the 3DGS rendering pipeline, identifying the alpha blending stage within the tile-based rasterizer as causing a significant performance bottleneck. Based on our findings, we discuss several future research directions aiming to inspire continued exploration within this burgeoning application domain.**

***Index Terms*—3D gaussian splatting, characterization, novel view synthesis, rendering, spatial computing.**

I. Introduction

3D RENDERING is widely utilized across various domains of computer graphics, including visual effects as well as applications in 3D reconstruction and extended reality. Among various 3D rendering techniques, Neural Radiance Fields (NeRF) [1], which leverages deep neural networks (DNNs), has so far been regarded as the state-of-the-art solution for generating images from arbitrary perspectives of a 3D model or scene. Despite its superior rendering quality, the high computation overheads of NeRF remained as a significant challenge in its deployment for latency-critical application domains.

Recently, a novel radiance field rendering technique called *3D Gaussian Splatting* (3DGS) [2] has emerged as a new state-of-the-art solution for 3D rendering, taking over the research community by storm. 3DGS is not based on computation-intensive DNNs, so it offers significant reduction in latency while not sacrificing the rendered image quality. Given 3DGS's impact on both academia and industry and the anticipated promise of its vast capabilities in various spatial computing applications, there is a clear need for a detailed workload characterization of 3DGS from a system's perspective. In this work, we conduct a comprehensive characterization of the end-to-end 3DGS rendering pipeline, identifying some of its critical system-level bottlenecks.

Received 27 August 2024; revised 19 October 2024; accepted 17 November 2024. Date of publication 21 November 2024; date of current version 17 January 2025. This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP), in part by the Korea government (MSIT) under Grant RS-2024-00438851, (SW Starlab) High Performance Privacy-preserving Machine Learning System and System Software and under Grant RS-2024-00402898, Simulation-based High-speed/High-Accuracy Data Center Workload/System Analysis Platform, and in part by the SNU-SK Hynix Solution Research Center (S3RC). The EDA tool was supported by the IC Design Education Center (IDEC), Korea. *(Corresponding author: Minsoo Rhu.)*

The authors are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: jiwon.lee@kaist.ac.kr; yunjae408@kaist.ac.kr; youngeun.kwon0405@gmail.com; mrhu@kaist.ac.kr).

Digital Object Identifier 10.1109/LCA.2024.3504579

II. Background and Motivation

A. Overview of the 3D Gaussian Splatting Rendering Pipeline

Components of 3D Gaussian in 3DGS: A 3D Gaussian in 3DGS is defined by four parameters: position μ, transformation details $\mathbf{\Sigma}$, color (c), and opacity α. The position is specified as $\langle x, y, z \rangle$, while $\mathbf{\Sigma}$ includes translation, rotation, and scaling. Color (c) and opacity α are key in determining pixel values during the alpha blending stage.

Fig. 1 provides an overview of a typical 3DGS rendering pipeline, which is implemented as tile-based rasterizer in the reference 3DGS paper [2]. Below we detail each of its major processing stages.

⓪ **Frustum Culling:** After all the trained Gaussian parameters are loaded in memory, redundant Gaussians are filtered out during this stage. Specifically, only those Gaussians visible from a given view frustum are selected by projecting lines from the camera origin to the view-frustum edges.

① **3D-to-2D Splatting:** The projection of 3D Gaussians into a 2D image screen involves computing the covariance projection function, $\mathbf{\Sigma}' = \mathbf{JW\Sigma W^T J^T}$, where J and W represent the Jacobian for camera-to-image projection, and the world-to-camera perspective transform matrix, respectively. $\mathbf{\Sigma}'$ is a covariance matrix in the 2D image coordinate system.

② **Tiling and** ③ **Indexing:** The 3D Gaussians are transformed into 2D Gaussians, known as splats, and are mapped to the 2D image screen with each retaining its own view-space depth information as index. The screen is divided into pixel tiles, the number of which depends on the thread-block size employed within the CUDA kernel used for parallel processing, the default size of which is 16×16.

④ **Duplicating Gaussians with Dictionary Keys:** Splats, each with depth information as an index, are distributed across multiple tiles. To facilitate parallel processing, these splats are duplicated based on the number of tiles overlapped by each splat. Subsequently, a dictionary key is assigned to each tile-splat pair, encoding the view-space depth of each splat and the overlapped tile index, resulting in a 64-bit-long key structure.

⑤ **Sort by Keys:** To ensure the correct processing order for alpha blending, the splats must be sorted based on depth. This sorting process is performed in parallel using a radix sort algorithm, one which is implemented using NVIDIA's CUDA CUB library. After sorting, lists of Gaussians per tile are generated by identifying the start and end of ranges in the array of sorted Gaussians within the same tile ID.

⑥ **Alpha Blending:** The alpha blending stage consists of the following four steps as described in Algorithm 1.

1) **Preparation:** Initially, the kernel initializes one thread-block per tile, verifies the validity of pixel ranges within each tile, and allocates per-tile shared memory as needed. Subsequently, each thread-block loads the Gaussian data packets into shared memory.

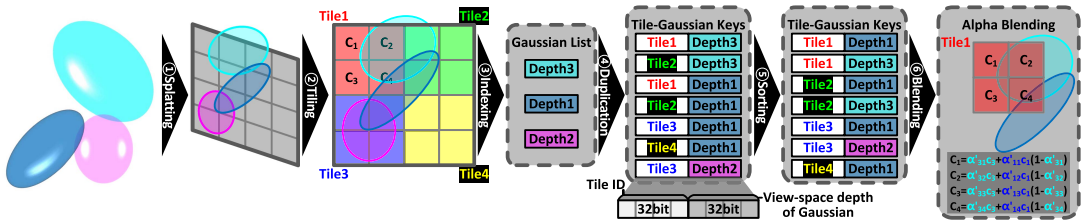

Fig. 1. High-level overview of the 3DGS rendering pipeline.

Algorithm 1: Depth-in-Order Alpha Blending Algorithm.

(X, Y): the current tile range
$\mathbf{p}(p_x, p_y)$: current pixel coordinate
$\mathbf{G_p}$: the list of overlapped Gaussians at $\mathbf{p}$
$\mathbf{G_i}$: ith Gaussian overlapped at $\mathbf{p}$
$\mathbf{o_i}(o_{ix}, o_{iy}, o_{iz}, o_{iw})$: ith Gaussian's conic opacity at (x,y,z)
α_i: ith Gaussian's 2D splat opacity
c_i: ith Gaussian's color
T_i: ith Gaussian's accumulated transmittance, T_0=1
C_i: $\mathbf{p}$'s color after blending ith Gaussian,
$C_0 = (R, G, B) = (0, 0, 0)$
ϵ_T: threshold for early termination by evaluating T_i

1: Identify current tile and pixel range ▷ Preparation
2: Allocate shared memory
3: **for p in** (X, Y) **do** ▷ Outer loop
4: Fetch Gaussian data from global to shared memory
5: **while** $\mathbf{G_i} \in \mathbf{G_p}$ **and** $T_i < \epsilon_T$ **do** ▷ Inner loop
6: $power = -0.5(o_{ix}x^2 + o_{iz}y^2 + 2o_{iy}xy)$ ▷ Compute exponent
7: $\alpha_i = o_{iw} \times \exp(power)$ ▷ Compute opacity
8: $T_i = T_{i-1} \times (1 - \alpha_i)$
9: $C_i = C_{i-1} + c_i\alpha_i T_i$ ▷ Compute color
10: **end while**
11: **end for**

2) **Exponent computation:** While iterating over the batch, the exponent value is computed from the conic matrix of 3D Gaussian by the surface splatting algorithm to calculate the opacity value in the subsequent step.
3) **Opacity computation:** By multiplying the fetched Gaussian opacity value by its exponential falloff from the mean, the opacity α value is computed.
4) **Color computation:** Lastly, the pixel's final color is derived and written to the image frame as rendering data.

B. Key Challenges and Motivation

Application domains like AR, VR, or SLAM require real-time rendering over power-limited environments like mobile/edge systems. To provide a satisfying end-user experience, however, there exists a certain level of SLA (service level agreement) latency, throughput, and image resolution requirements to be met. While high-end GPUs can effectively accelerate the 3DGS rendering pipeline's inference demands, the high power consumption associated with these power-hungry GPUs motivate the exploration of alternative domain-specific architectures tailored for 3DGS. As we delve deeper in the following section, the end-to-end latency of 3DGS varies depending on the number of Gaussians coming into the rendered view, presenting a significant challenge in meeting SLA under a tight compute and power budget. This mismatch can lead to low efficiency, utilization, and ultimately, performance degradation. In the remainder of our study, we aim to identify the performance bottlenecks of 3DGS by characterizing its end-to-end rendering pipeline, with a specific focus on the alpha blending stage, and subsequently, explore potential strategies for acceleration.

III. Methodology

A. Experimental Setup & Datasets

Datasets: We utilized 13 real-world scenes from Tanks & Temples [3], Mip-NeRF360 [4], and DeepBlending [5] datasets, and 8 synthetic objects from the Blender dataset [1], consistent with the original 3DGS evaluation setup [2]. All models were trained for 30,000 iterations.

Hardware & software: Our evaluation platform comprises an NVIDIA A100 GPU with 40 GB of HBM2 memory, hosted by an AMD EPYC 7502 CPU with 512 GB of DRAM. We also characterize 3DGS over a mobile edge computing platform using NVIDIA's Xavier AGX SoC with 16 GB of LPDDR4 memory. The software platform includes CUDA version 11.8 and PyTorch version 2.1. For workload characterization, we employed Nsight Systems and Nsight Compute, utilizing the NVTX (NVIDIA Tools Extension SDK) library as well as *nvidia-smi* for A100 and *tegrastats* for Xavier AGX. Additionally, for the implementation of 3DGS [2] and Instant-NGP [6], we utilized the official open-source codes provided by the authors of 3DGS and Instant-NGP, respectively.

B. NeRF-Based Approaches' Workload

Before the rise of 3DGS, NeRF-based methods were predominantly used for novel view synthesis. Among these, Instant-NGP [6] stands out for its high rendering quality and speed. The main bottlenecks in NeRF-based approaches have been identified as MLP operations and parametric encoding, as highlighted by previous research aimed at accelerating Instant-NGP [7], [8]. As the bottlenecks of NeRF is well-explored in the cited references, this paper focuses on profiling 3DGS.

IV. Characterization

A. Latency Breakdown

End-to-end 3DGS rendering pipeline: To identify bottlenecks in the 3DGS rendering pipeline, we measured latency for each component (2). There are two key observations we make from this evaluation. First, there exists a strong correlation between the end-to-end latency and the number of Gaussians (i.e., the trained 3DGS model size which is directly dependent on the number of Gaussians). 3DGS models trained from synthetic datasets (e.g., SyntheticNeRF) typically have fewer number of Gaussians vs. real-world scenes because it lacks background

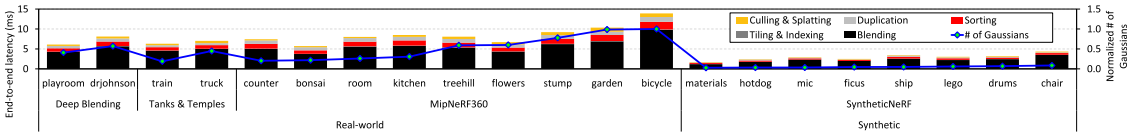

Fig. 2. End-to-end latency breakdown of 3DGS rendering pipeline measured over NVIDIA's A100 and its correlation with the number of Gaussians in the trained 3DGS model. The left axis indicates averaged end-to-end latency of the 3DGS rendering pipeline per test dataset. The right axis represents the normalized number of Gaussian points (equivalent the number of point clouds derived from image features), which constitutes the trained 3DGS model.

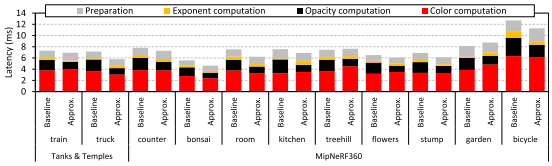

Fig. 3. Latency breakdown of the alpha blending stage measured on A100. As discussed later in Section V, the opacity computation stage heavily utilizes the single-precision exp function (line 7 in Algorithm 1), which accounts for a significant fraction of the latency to derive opacity. We observe that switching this function to an *approximate* version of the original exp function (i.e., $\_\_expf$, a function that is provided as part of the NVIDIA Math library) helps reduce the latency to derive opacity while incurring almost no degradation in the quality of 3DGS rendering.

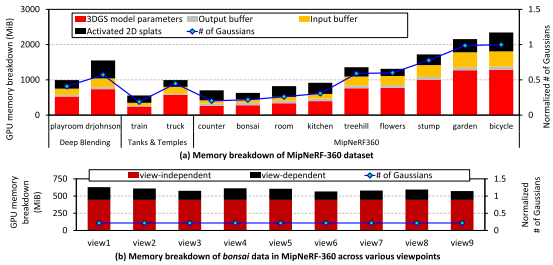

Fig. 4. Memory breakdown of 3DGS rendering pipeline (a) based on its usage and (b) as a function of the viewpoint.

information, leading to a reduced number of point clouds derived from image features to initialize 3D Gaussians. Second, the alpha blending stage is the most significant bottleneck, accounting for 72% of total inference time.

Execution time of alpha blending stage: Given that the alpha blending stage dominates the end-to-end execution time of 3DGS rendering, we delve deeper into this bottleneck stage by breaking down its execution time into the four key steps outlined in Algorithm 1: (1) preparation, and the computation of (2) exponent, (3) opacity, and (4) color. Derivation of opacity and color accounts for the majority of alpha blending time, as shown in Fig. 3, making these two steps a primary target for acceleration.

B. Memory Consumption

Overall memory consumption: To quantify the memory usage during 3DGS rendering, we used *nvidia-smi* to monitor the change in memory usage over 20 ms intervals. Across Mip-NeRF360's nine test datasets, the average/maximum memory usage during 3DGS rendering was 1,235 MB/2,344 MB, respectively. Given VR devices like Meta Quest 2 have 6 GB total memory, efficient memory solutions during 3DGS rendering will be crucial for mobile/edge deployment.

Memory breakdown (per usage): In Fig. 4(a), we breakdown the overall memory consumption based on its usage. The *3DGS model parameters*, including the trained point cloud with the four components of 3DGS, dominate memory consumption. The *input buffer* and *output buffer* are temporary data buffers used to store the pre-processed 3D Gaussian parameters, whose size is proportional to the number of points included in the trained 3DGS model, and the image frame data determined by the size of (height $\times$ width) of the output image frame, respectively. The *activated 2D splats* represent the portion of actual 2D Gaussian splat data used in the main alpha blending algorithm. Upon careful examination of the *3DGS model parameters* portion, we observe that the internal composition of these parameters mirrors that of the trained model's point cloud data file. This data file consists of 3 $\langle x, y, z \rangle$ coordinates, 3 scale parameters, 3 rotation parameters, 48 color feature parameters for spherical harmonics (SH) computation, and a single opacity value, all of which are stored as 4-byte single-precision floats. The color feature parameters dominate the memory usage, suggesting that significant reductions in the memory footprint of the trained 3DGS models could be achieved through efficient storage of color feature data. Future accelerators targeting 3DGS can potentially incorporate this insight to design a memory-efficient 3DGS rendering pipeline.

Memory breakdown (as a function of viewpoint): We show the memory usage of 3DGS rendering when the viewpoint is changed in Fig. 4(b). Specifically, we evaluate 9 different perspectives of the *bonsai* test dataset in Mip-NeRF360 and categorize its memory usage based on *view-independent* and *view-dependent* portions. The *view-independent* part encompasses the *3DGS model parameters*, *input buffer*, and *output buffer*, while the *activated 2D splats* constitute the *view-dependent* part (as discussed in Fig. 4(a)). Because the size of *3DGS model parameters*, *input buffer* and *output buffer* is the function of the number of Gaussians and the output image frame size, these *view-independent* memory allocation size can be derived deterministically, regardless of the camera viewpoint. Conversely, the memory consumption of the *view-dependent* part exhibits variations depending on the viewpoint of the test dataset, making its memory footprint unpredictable.

C. Analysis on Edge Computing Environment

Latency breakdown: Fig. 5 shows a breakdown of 3DGS's end-to-end latency over Xavier AGX, exhibiting similar characteristics to those observed with a server-class A100 GPU (Fig. 2). However, the overall latency on the Xavier AGX is more than ten times longer than on the A100, which is expected given the significant performance disparity between Xavier AGX and A100.

Power consumption breakdown: Applications such as XR and SLAM, which can benefit from 3DGS, often operate over portable, battery-powered edge computing devices, rendering low power consumption a crucial design factor. We measure the power consumption of 3DGS rendering over Xavier AGX using NVIDIA's *tegrastats* utility. Since the performance mode of Xavier AGX is set to *MAXN*, where all CPU & GPU cores operates, the overall power consumption is constantly fixed to 30 W, resulting in the total dissipated energy proportional to the end-to-end latency (Fig. 7). On average, the power consumption

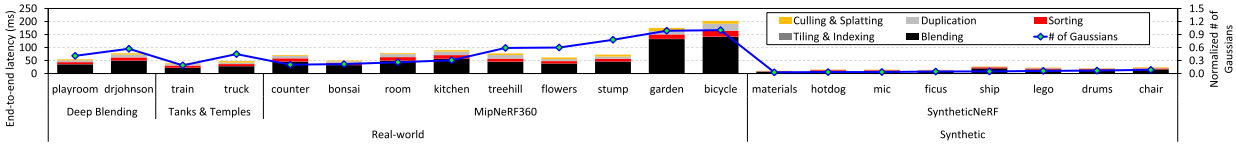

Fig. 5. Breakdown of end-to-end latency of 3DGS rendering pipeline, measured over NVIDIA's Xavier AGX edge computing platform.

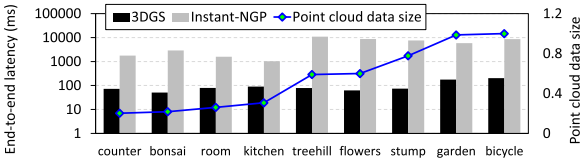

Fig. 6. End-to-end rendering latency comparison of 3DGS vs. Instant-NGP, evaluated on Mip-NeRF360 dataset with Xavier AGX.

Fig. 7. Comparison of total dissipated energy between 3DGS and Instant-NGP to render a single image, evaluated on Mip-NeRF360 real-world dataset with Xavier AGX platform. The evaluation of energy consumption over A100 exhibited similar trends and we omit the results for brevity.

distribution was measured as follows: 55.5% for the GPU, 5.1% for the CPU, 8.1% for DRAM (LPDDR4), 15.5% for peripheral I/O and power rails, and 15.7% for other embedded processor components within the SoC.

V. Future Directions

Hardware/algorithm co-design for 3DGS: Our characterization revealed that alpha blending is a bottleneck for designing a robust 3DGS rendering pipeline. To address this challenge, we identify some promising research directions for accelerating this critical step, one of which we introduce below. As illustrated in Fig. 3, the opacity computation step of alpha blending requires frequent usage of the exponential function, which is implemented using a single-precision exp function in CUDA (*Baseline* in Fig. 3). We observe that simply switching this function to an *approximate* version of the baseline exp function (which is implemented as $\_\_expf$ in the NVIDIA Math library, denoted *Approx.* in Fig. 3) helps reduce the end-to-end latency of the 3DGS rendering pipeline by 8.2% without compromising the PSNR of the rendered image quality – a purely software-only optimization that introduces non-trivial performance improvements. These results highlight the potential for designing a 3DGS hardware accelerator co-designed with the 3DGS algorithm that provides substantial improvements in energy-efficiency. We leave the exploration of such research direction as future work.

SLA-aware 3DGS rendering pipeline: The end-to-end latency of 3DGS rendering is significantly impacted by the number of 3D Gaussians, in contrast to NeRF-based algorithms, which are less affected by dataset size. In Fig. 6, 3DGS exhibits greater latency fluctuations with increasing scene complexity, posing challenges for real-time applications with strict SLA requirements. While executing the 3DGS rendering pipeline on high-end/server-grade GPUs currently delivers sufficient performance for real-time operation, meeting throughput and power consumption requirements becomes increasingly difficult particularly on resource-constrained edge systems. On the other hand, Instant-NGP is relatively insensitive to scene complexity, but it requires more than ten times the latency compared to 3DGS to render a single image frame, resulting in significantly higher energy consumption (Fig. 7). To address these limitations, the goal of our future work is to explore optimizations like quantization and overlapping sub-stages to reduce latency and improve power efficiency, particularly for resource-constrained edge systems. Lowering precision in less critical stages can enhance throughput while reducing memory usage. Insights from Instant-NGP's insensitivity to scene complexity will also guide the development of SLA-aware architectural solutions to ensure balanced rendering quality, performance, and power consumption.

VI. Related Work

3DGS was introduced in August 2023 [2] and has quickly superseded conventional NeRF-based radiance field rendering algorithms, becoming the state-of-the-art solution for 3D rendering. Despite its quick emergence, there is little to no prior work investigating 3DGS from a computer system's perspective. Instead, numerous research endeavors have aimed at accelerating the training [7], and inference [8] of NeRF-based variants, particularly targeting Instant-NGP [6], a state-of-the-art NeRF algorithm.

VII. Conclusion

To the best of our knowledge, this work is the first to provide a quantitative analysis on 3DGS, presenting invaluable insights to computer architects for this emerging algorithm. Future work entails the development of acceleration solutions for end-to-end 3DGS rendering utilizing approximate computing with a focus on minimizing the latency variation dependent upon the trained model size. These solutions should be tailored to leverage the key findings of our study effectively.

References

[1] B. Mildenhall et al., "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 405–421.
[2] B. Kerbl et al., "3D Gaussian splatting for real-time radiance field rendering," *ACM Trans. Graph.*, vol. 42, no. 4, 2023, Art. no. 139.
[3] A. Knapitsch et al., "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Trans. Graph.*, vol. 36, no. 4, 2017, Art. no. 78.
[4] J. T. Barron et al., "Mip-NeRF 360: Unbounded anti-aliased neural radiance fields," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 5460–5469.
[5] P. Hedman et al., "Deepblending for free-viewpoint image-based rendering," *ACM Trans. Graph.*, vol. 37, no. 6, 2018, Art. no. 257.
[6] T. Müller et al., "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, 2022, Art. no. 102.
[7] S. Li et al., "Instant-3D: Instant neural radiance field training towards on-device AR/VR 3D reconstruction," in *Proc. 50th Annu. Int. Symp. Comput. Architecture*, 2023, Art. no. 6.
[8] J. Lee et al., "NeuRex: A case for neural rendering acceleration," in *Proc. Annu. Int. Symp. Comput. Architecture*, 2023, Art. no. 21.