No Redundancy, No Stall: Lightweight Streaming 3D Gaussian Splatting for Real-time Rendering

Linye Wei[1,2], Jiajun Tang[3,4], Fan Fei[3,4], Boxin Shi[3,4], Runsheng Wang[2,5,6], Meng Li[1,2,6]*
[1]Institute for Artificial Intelligence, Peking University, Beijing, China
[2]School of Integrated Circuits, Peking University, Beijing, China
[3] State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University
[4]National Engineering Research Center of Visual Technology, School of Computer Science, Peking University
[5]Institute of Electronic Design Automation, Peking University, Wuxi, China
[6]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China
meng.li@pku.edu.cn

***Abstract*—3D Gaussian Splatting (3DGS) enables high-quality rendering of 3D scenes and is getting increasing adoption in domains like autonomous driving and embodied intelligence. However, 3DGS still faces major efficiency challenges when faced with high frame rate requirements and resource-constrained edge deployment. To enable efficient 3DGS, in this paper, we propose LS-Gaussian, an algorithm/hardware co-design framework for lightweight streaming 3D rendering. LS-Gaussian is motivated by the core observation that 3DGS suffers from substantial computation redundancy and stalls. On one hand, in practical scenarios, high-frame-rate 3DGS is often applied in settings where a camera observes and renders the same scene continuously but from slightly different viewpoints. Therefore, instead of rendering each frame separately, LS-Gaussian proposes a viewpoint transformation algorithm that leverages inter-frame continuity for efficient sparse rendering. On the other hand, as different tiles within an image are rendered in parallel but have imbalanced workloads, frequent hardware stalls also slow down the rendering process. LS-Gaussian predicts the workload for each tile based on viewpoint transformation to enable more balanced parallel computation and co-designs a customized 3DGS accelerator to support the workload-aware mapping in real-time. Experimental results demonstrate that LS-Gaussian achieves 5.41× speedup over the edge GPU baseline on average and up to 17.3× speedup with the customized accelerator, while incurring only minimal visual quality degradation.**

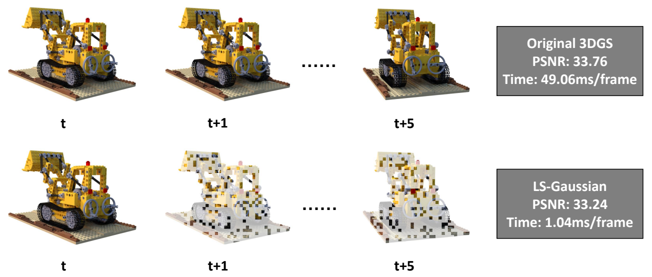


Fig. 1: Rendering with 3DGS on Jetson AGX Orin and LS-Gaussian. LS-Gaussian is accelerated by sparse rendering between consecutive frames and only needs to fully render one in every 6 frames.

I. Introduction

Volume rendering is a fundamental technique for reconstructing and synthesizing novel views of 3D scenes, playing a crucial role in applications such as augmented/virtual reality (AR/VR) [1], [2] and autonomous driving [3], [4]. With the emergence of embodied intelligence [5]–[7] and world models [8], [9], 3D representation has become an essential modality for bridging the physical and virtual worlds. This shift imposes stringent requirements on both the quality and efficiency. However, traditional methods such as Neural Radiance Fields (NeRF) [10], [11] are inadequate to meet these increasing demands, necessitating more efficient and scalable solutions.

3D Gaussian Splatting (3DGS) [12]–[14] models scenes as collections of Gaussian ellipsoids with varying shapes, colors, and opacities, synthesizing novel views through projection and accumulation of Gaussians onto the image plane. With explicit representations and tile-based parallel rasterization, 3DGS outperforms NeRF-based methods [15]–[17] in both quality and rendering speed, gaining significant attention since its introduction. However, as scene scales expand in practical applications [18], [19], 3DGS must process millions or more Gaussians, incurring substantial memory and computational overhead. On resource-constrained edge platforms, such as the Jetson AGX Orin, which are critical for many real-world tasks, 3DGS still struggles to achieve real-time rendering at 90 FPS [20].

Recent acceleration efforts focus on reducing Gaussian primitives and model size via quantization [21], [22], pruning [23]–[25], and level-of-detail (LOD) hierarchical rendering [26], [27]. However, these methods often degrade rendering quality and require costly retraining. Other approaches [20], [28]–[30] leverage hardware architecture innovations to enhance rendering efficiency but lack holistic pipeline optimization, limiting acceleration gains. More importantly, they overlook the high similarity between consecutive frames, missing opportunities to leverage prior frames for real-time rendering.

In this paper, we present a detailed analysis of bottlenecks across different stages of the 3DGS rendering pipeline, revealing inefficiencies caused by both computational redundancy and hardware stalls. The redundancy arises primarily from two sources. On one hand, 3DGS is often deployed in scenarios where an observer renders a scene from slightly shifting viewpoints across consecutive frames. This results in substantial inter-frame similarity, making it unnecessary to fully re-render every frame. On the other hand, coarse intra-frame intersection tests introduce a large number of invalid Gaussian-tile pairs into the rendering pipeline, further exacerbating redundant computation. In parallel, hardware stalls are mainly attributed to inter-block idling and intra-block bubbles, which stem from imbalanced and unpredictable tile workloads. The naive assignment of tiles to rendering blocks fails to achieve a balanced load distribution, leading to significant underutilization of hardware resources.

Motivated by these observations, we propose LS-Gaussian, a

This work was supported in part by NSFC under Grant 62495102, Grant 92464104, Grant 62125401, and Grant 62136001, in part by Beijing Outstanding Young Scientist Program under Grant JWZQ20240101004, in part by Beijing Municipal Science and Technology Program under Grant Z241100004224015, and in part by 111 Project under Grant B18001.

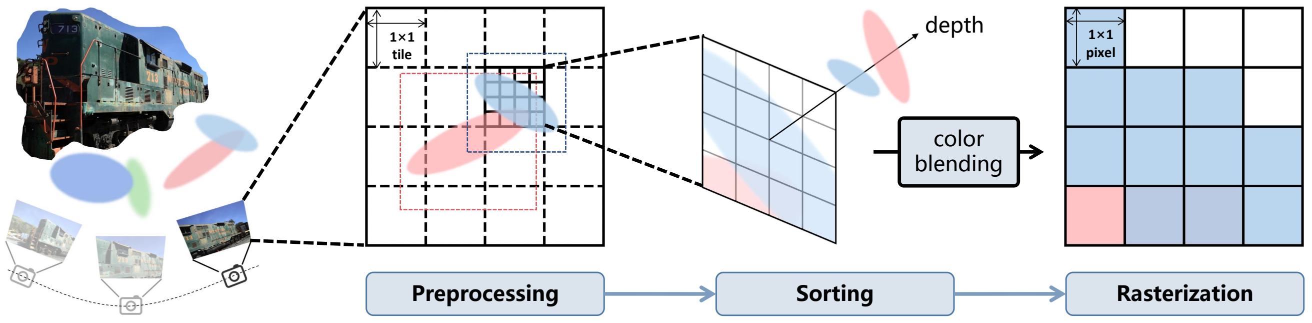


Fig. 2: Overview of the 3DGS pipeline.

3DGS pipeline optimized for real-time rendering without retraining, leveraging frame continuity, as illustrated in Fig. 1. By refining the algorithm from inter-frame level to intra-frame level, we achieve redundancy-free, lightweight rendering. Additionally, we design a load-balanced accelerator architecture tailored for 3DGS, featuring a streaming pipeline that allows continuous processing across hardware modules without requiring global synchronization. This design improves hardware utilization and further enhances acceleration. Our main contributions can be summarized as follows:

- We conduct a comprehensive bottleneck analysis of the entire 3DGS pipeline, identifying two sources of redundancy and two types of stalls that limit rendering performance.
- To address these inefficiencies, we propose Lightweight Streaming 3DGS (LS-Gaussian), which optimizes the pipeline from both algorithmic and architectural perspectives. Our training-free approach enables seamless integration with existing methods.
- Extensive experiments across multiple datasets demonstrate that LS-Gaussian achieves an average 5.41× speedup on Jetson AGX Orin GPU. With dedicated hardware support, LS-Gaussian further achieves a 17.3× speedup while maintaining high quality.

II. Background

A. Preliminaries for 3D Gaussian Splatting

3DGS represents a 3D scene using Gaussian ellipsoids, learned from a sparse point cloud generated by Structure from Motion (SfM) [31]. Each Gaussian is defined by its position, covariance matrix, opacity, and spherical harmonic coefficients, which encode its appearance color. During inference, the color of each pixel on the camera plane is computed by accumulating contributions from Gaussians projected onto that point. As illustrated in Fig. 2, novel view synthesis in 3DGS consists of three stages:

Preprocessing. Gaussians outside the camera frustum are first culled and the remaining ones are projected onto the camera plane, which is subdivided into 16×16-pixel tiles. An axis-aligned bounding box (AABB) intersection test determines the tiles intersected by each Gaussian. It enables all pixels within the same tile to share common Gaussians, effectively leveraging the parallel computing capabilities of GPU streaming multiprocessors (SMs) during following steps.

Sorting. To accurately resolve occlusions from the viewpoint, Gaussians within each tile are sorted in depth order. This sorted sequence dictates the processing order during rasterization, ensuring correct color accumulation and blending.

Rasterization. SMs are organized into 16×16-thread blocks, where each tile is mapped to a unique block and each pixel within the tile is assigned to a distinct thread. During parallel rendering, threads within a block process the Gaussians covering the tile in Single Instruction Multiple Threads (SIMT) manner, following the sorted depth order. Each thread evaluates the density α_i of Gaussian i with opacity o_i:

$$\alpha_i = o_i e^{-\frac{1}{2}\left(P-\mu_i'\right)^\top \Sigma_i'^{-1}\left(P-\mu_i'\right)}, \tag{1}$$

where P represents the pixel coordinate, μ_i', Σ_i' are the position and 2D covariance matrix of the projected Gaussian, respectively. If the density exceeds a predefined threshold ($\frac{1}{255}$), the Gaussian color c_i accumulates, and the pixel color C can be subsequently calculated through the volume rendering (α-blending) process:

$$C = \sum_{i\in N} c_i\alpha_i T_i = \sum_{i\in N} c_i\alpha_i \prod_{j=1}^{i-1}\left(1-\alpha_j\right). \tag{2}$$

Once the cumulative transmittance T_i drops below another threshold (10^{-4}), the pixel is considered fully rendered and thread execution is terminated, which is known as early stopping [28].

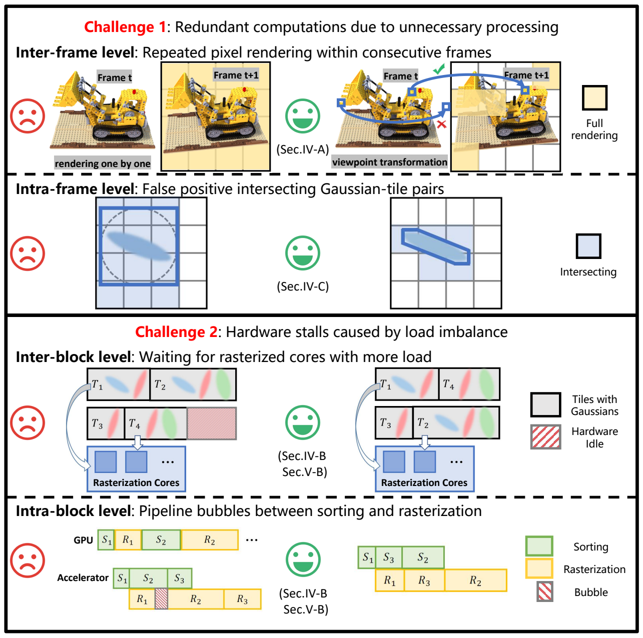
B. Acceleration of 3D Gaussian Splatting

Several studies have highlighted the need to accelerate 3DGS. A common approach is to reduce the number of Gaussians and parameter size. Early efforts [23]–[25] explored quantization and pruning techniques, selectively removing Gaussians based on rendering sensitivity. Recently, the demand for reconstruction of city-scale scenes drives the adoption of level-of-detail (LOD) [20], [26], [27], which dynamically adjusts the Gaussian resolution based on depth and visual attention. However, these methods inevitably degrade reconstruction quality and rely on retraining. As 3DGS has become a core spatial representation, numerous studies have rapidly emerged to adapt it to new application scenarios. Retraining-based approaches require substantial modifications to accommodate different reconstruction paradigms, limiting their general applicability.

Some studies have explored accelerating 3DGS without retraining. Pixel and Gaussian reuse [32], [33] exploit the similarity between adjacent frames to alleviate computational and memory constraints, but fail to account for the tile-based rendering characteristics of 3DGS, resulting in limited speedup. Preprocessing techniques [28], [30], [34], [35] improve the Gaussian-tile intersection accuracy, reducing unnecessary sorting and rendering, but often incur high computational costs due to complex analytical geometry. Meanwhile, GPU enhancements [36], [37] and streaming designs [28], [29] aim to mitigate mutual stalls between hardware units, but lack fine-grained evaluation of workload imbalance. In contrast, we systematically address bottlenecks across the entire 3DGS pipeline, minimizing unnecessary computation while maximizing hardware utilization.

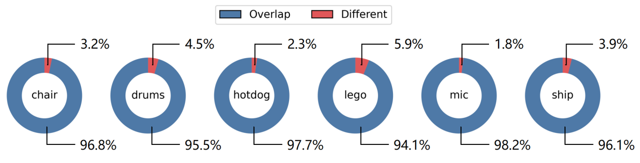
III. Motivation

To further investigate the performance bottlenecks in the 3DGS pipeline, we perform a comprehensive study of rendering speed and

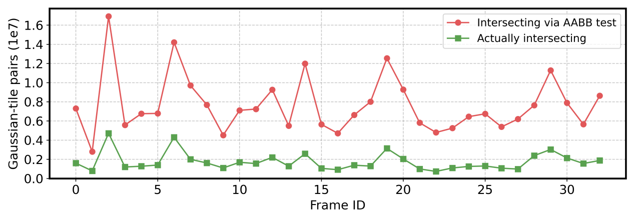


Challenge 1: Redundant computations due to unnecessary processing
Inter-frame level: Repeated pixel rendering within consecutive frames
Frame t
Frame t+1
rendering one by one
(Sec.IV-A)
viewpoint transformation
Full rendering
Intra-frame level: False positive intersecting Gaussian-tile pairs
(Sec.IV-C)
Intersecting
Challenge 2: Hardware stalls caused by load imbalance
Inter-block level: Waiting for rasterized cores with more load
Rasterization Cores
(Sec.IV-B Sec.V-B)
Tiles with Gaussians
Hardware Idle
Intra-block level: Pipeline bubbles between sorting and rasterization
GPU
Accelerator
Sorting
Rasterization
Bubble

Fig. 3: Algorithmic and hardware challenges of 3DGS.



(a)



(b)

Fig. 4: (a) Proportion of overlap pixels between consecutive frames on multiple scenes and (b) the intersecting Gaussian-tile pairs judged by AABB test of the original 3DGS and actual intersecting pairs in the "drjohnson" scene test set.

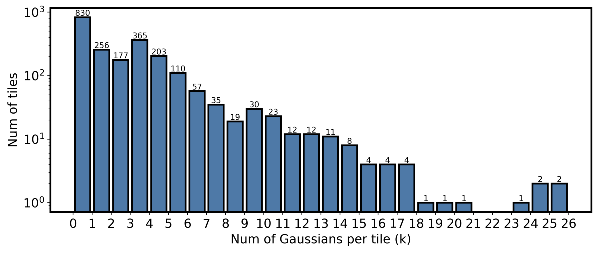


Fig. 5: Distribution of the covered Gaussian numbers on different tiles in a frame from the "train" scene.

hardware efficiency across multiple datasets. Our findings indicate that inefficiencies primarily arise from redundant computations due to unnecessary processing and hardware stalls caused by load imbalance, as shown in Fig. 3. Based on these insights, we present two key observations that motivate our subsequent work.

Observation 1: Redundant computation exists across multiple levels of 3DGS rendering. At the **inter-frame level**, real-time 3D rendering demands seamless transitions between frames, leading to substantial pixel overlap, especially in static or slowly changing regions, as illustrated in Fig. 4a. We emphasize that existing efforts [32] to handle viewpoint transformations in 3DGS remain inadequate. Since projected Gaussians are assigned to workloads at the tile level, naive pixel warping fails to reduce preprocessing and sorting costs. Moreover, rendering quality degrades noticeably as the number of consecutively transformed frames increases, due to the accumulation of interpolation errors. In addition, viewpoint transformations inherently provide depth priors that could be leveraged to predict early stopping positions, enabling more accurate tile load estimation for balanced task allocation. However, this potential remains largely unexploited.

At the **intra-frame level**, the original preprocessing stage overestimates the number of tiles covered by each Gaussian and increases invalid Gaussian-tile pairs to be rendered. However, as shown in Fig. 4b, this coarse bounding box approximation associates each tile with numerous Gaussians that do not actually contribute to rendering, significantly increasing the overhead in both the sorting and rasterization stages. Analytical geometry-based methods provides a more accurate intersection test but introduce significant overhead in the preprocessing stage.

Observation 2: Load imbalance causes inefficient hardware utilization. As mentioned in Sec. II, tiles are mapped to streaming multiprocessor (SM) blocks for parallel rendering. However, as illustrated in Fig. 5, where image tiles are grouped by the number of covered Gaussians, variations in scene complexity cause the per-tile Gaussian count to vary by more than an order of magnitude, leading to severe load imbalance.

At the **inter-block level**, this imbalance causes lightly loaded blocks to finish rasterization early but remain idle, waiting for heavily loaded blocks to complete rendering. On edge devices, additional processing waves due to limited compute resources help mitigate this imbalance by allowing underutilized blocks to render more tiles. However, sparse rendering which exploit frame-to-frame similarity reduce total workload and wave count, making inter-block waiting a critical bottleneck again. At the **intra-block level**, traditional GPU platforms perform sorting and rendering sequentially on SMs. Recent works such as GSCore [30] enable parallelism between sorting and rasterization across tiles by decoupling these stages into dedicated hardware units, thereby reducing latency across them. However, the uncertainty in workload distribution results in potential bubbles in the rasterization unit, waiting for potentially prolonged sorting.

IV. Algorithm Optimization for Lightweight 3DGS

Based on the insights from above analysis, we propose a series of algorithmic optimizations for lightweight 3DGS that eliminate computational redundancy at inter-frame and intra-frame levels. First, We introduce a novel sparse rendering strategy specifically designed for 3DGS (Sec. IV-A), utilizing tile warping instead of the traditional pixel warping, thereby better aligning with the tile-based architecture of the pipeline. By leveraging depth map from viewpoint transformations, we then predict the locations where early stopping is likely to occur (Sec. IV-B), effectively estimating the load of each tile.

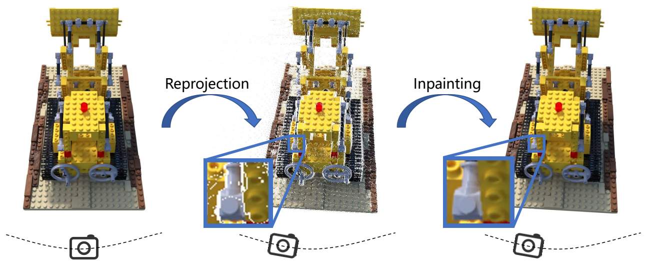


Fig. 6: An example of viewpoint transformation and sparse rendering between consecutive frames. The reference frame is reprojected to the target viewpoint, and then inpaint the pixels that have no reprojection source.

Additionally, we adopt a two-stage strategy to accurately determine the intersections between Gaussians and tiles (Sec. IV-C).

A. *Tile Warpping-based Sparse Rendering (TWSR)*

In real-time rendering scenarios, the high similarity between consecutive frames motivates the reuse of pixels to reduce redundant computations. As illustrated in Fig. 6, given a reference frame, we apply a viewpoint transformation to determine which pixels can be reused in the next frame. Specifically, pixels from the reference frame are first back-projected into 3D space using scene depth and camera pose, forming a point cloud. These 3D points are then transformed according to the target frame's camera pose and reprojected onto its image plane. Due to viewpoint changes and resulting occlusions, the reprojected target frame typically contains missing pixels, which need to be filled through rendering.

Pixel warping (PW). Existing methods [16], [32] often use a NeRF-style pixel warpping-based sparse rendering (PWSR) strategy, where only the missing pixels in the target frame are filled. However, we observe that this simple pixel warping approach conflicts with the 3DGS rendering pipeline. Unlike NeRF, where pixels are rendered independently, 3DGS organizes rendering by tiles, with all pixels in a tile sharing the same set of Gaussians. As a result, preprocessing and sorting cannot be skipped unless no pixel within a tile requires rendering. While rasterization remains the dominant factor in overall rendering time, sparse rendering reduces its workload, making the cost of preprocessing and sorting more pronounced. Moreover, unlike previous works that assume known ground truth depth, we estimate the depth in real-time by computing an opacity-weighted sum over the depths of the contributing Gaussians. While this estimation is generally accurate, it inevitably introduces minor errors. These small errors tend to be amplified in regions with limited pixel reuse, where view warping is less effective and missing pixels are more frequent. In such areas, the projected pixel positions deviate from the actual rendering results, ultimately degrading the final image quality.

Tile warping (TW). To address the issues mentioned above, we propose a tile-based inpainting strategy. For tiles with a small number of missing pixels (empirically set to less than one-sixth of the total pixels in the tile), we observe that these typically correspond to regions with smooth depth variation and similar color distributions. In such cases, we directly interpolate the remaining pixels, bypassing not only the rasterization stage but also the preprocessing and sorting. On the other hand, for tiles with a large number of missing pixels, we re-render the entire tile to ensure high visual fidelity. As shown in Fig. 7, our tile-level warping consistently achieves higher PSNR values compared to pixel-based warping approaches.

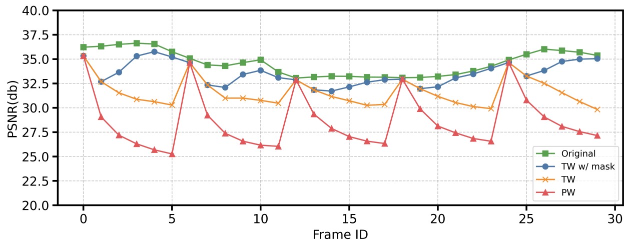


Fig. 7: Comparison of image quality under different inpainting strategies for the "chair" scene in the Synthetic-NeRF dataset.

No cumulative error mask (TW w/ mask). Unfortunately, as the number of consecutive viewpoint transformations increases, we observe a noticeable degradation in image quality. This is primarily due to the accumulation of interpolation errors across frames. The more transformation rounds applied, the more severe the quality deterioration becomes. To mitigate this issue, we introduce a masking mechanism for interpolated pixels. These pixels are treated as "blank" during subsequent view projections and are excluded from contributing to the next frame. As shown in Fig. 7, this masking strategy significantly enhances rendering quality compared to direct inpainting. Remarkably, the quality continues to improve with more consecutive frame transformations. This improvement is attributed to the increased number of missing pixels introduced by the mask in the target frame. According to our tile-based inpainting policy, this leads to a little more tiles being fully re-rendered, thereby striking a balance between efficiency and image quality.

B. *Depth Prediction for Early Stopping (DPES)*

As mentioned in Sec. III, some tiles are covered by thousands of Gaussians, yet a significant portion of these Gaussians does not contribute to the color of the pixels in these tiles due to early stopping. As a result, the total number of overlapping Gaussians computed during preprocessing does not accurately reflect the actual workload per tile. To better estimate rendering cost, it is necessary to predict the early stopping position, which determines the number of Gaussians that are effectively traversed during rasterization. In a typical rendering pipeline, this truncation point is unpredictable and depends on the opacity accumulation of Gaussians at each pixel during the rasterization stage. However, by leveraging viewpoint transformations based on scene depth and camera pose, we can predict and address this issue more effectively.

We emphasize that viewpoint transformations not only enable target frames to inherit pixel colors from reference frames but also their corresponding depths. Building on this, we propose a method to predict the early stopping point for each tile, leveraging the truncated depth from the reference frame. During the initial rendering, we record the depth of each pixel in the reference frame, which corresponds to the depth of the last Gaussian after traversing all Gaussians or the depth if early stopping occurs. These depths are reprojected onto the new camera plane. We define the early stopping depth for each tile in the target frame as the maximum depth of all valid reprojected pixels within that tile. Any Gaussians beyond this depth will not be involved in sorting in the target frame. The complete viewpoint transformation process, including sparse rendering and truncated depth estimation, is outlined in Algo. 1.

The early stopping estimation can be utilized to predict the expected load of each tile after sorting, allowing for a more balanced distribution of the workload across rasterization units. This signifi-

Algorithm 1 Viewpoint Transformation Process

```
Input: A reference frame $F_{\text{ref}}$, scene depth map $D_{\text{ref}}$, truncated depth
    map $D^{\max}_{\text{ref}}$, reference viewpoint $V_{\text{ref}}$, target viewpoint $V_{\text{tgt}}$, re-
    rendering threshold $N_0$
Output: A target frame $F_{\text{tgt}}$ with a groups of tiles $T$, tile-level
    truncated depth map $D_{\text{T}}$
 1: $F_{\text{tgt}}, D_{\text{T}} \leftarrow \text{Initialize}()$
 2: $P_{\text{ref}}, P^{\max}_{\text{ref}} \leftarrow \text{ProjectTo3D}(F_{\text{ref}}, D_{\text{ref}}, D^{\max}_{\text{ref}})$
 3: $P_{\text{tgt}}, P^{\max}_{\text{tgt}} \leftarrow \text{ViewTransfer}(P_{\text{ref}}, P^{\max}_{\text{ref}})$
 4: $F_{\text{tgt}}, D^{\max}_{\text{tgt}} \leftarrow \text{Reproject}(P_{\text{tgt}}, P^{\max}_{\text{tgt}})$
 5: for $t \in T$ do
 6:     $N \leftarrow \text{ValidPixelSum}(F_{\text{tgt}}, t)$
 7:     if $N > N_0$ then
 8:         $\text{Interpolate}(F_{\text{tgt}}, t)$
 9:     else
10:         $D_{\text{T}} \leftarrow \text{Max}(D^{\max}_{\text{tgt}}, t)$
11:         $\text{Re-render}(F_{\text{tgt}}, t)$
12:     end if
13: end for
```

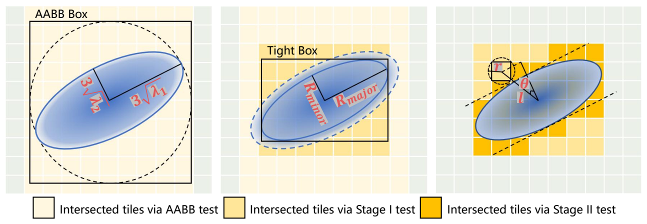

AABB Box
$3\sqrt{\lambda_2}$
$3\sqrt{\lambda_1}$
Tight Box
R_{minor}
R_{major}
r
θ
l
Intersected tiles via AABB test
Intersected tiles via Stage I test
Intersected tiles via Stage II test

Fig. 8: Two-stage accurate intersection test.

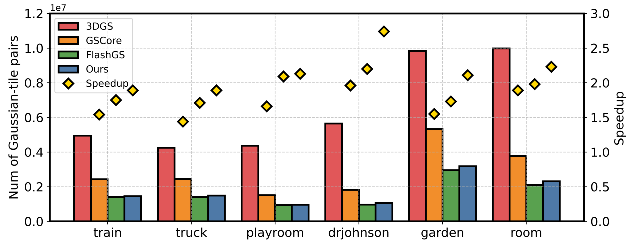


Fig. 9: Comparison of our two-stage accurate intersection test with previous works on the number of Gaussian-tile pairs and Speedup across multiple scenes.

cantly mitigates kernel idle caused by load imbalance, as discussed in the next section.

C. Two-stage Accurate Intersection Test (TAIT)

As mentioned in Sec. II, a simple AABB test is commonly used to determine which tiles are covered by each Gaussian. This approach is efficient for preprocessing, as it only requires the projected center and the semi-major axis length of the Gaussian. The bounding box is constructed as the circumscribed square of a circle centered at the Gaussian's projection, with a radius equal to the semi-major axis. All tiles intersecting with this square are considered to intersect with the Gaussian. However, as shown in Fig. 8, this approximation leads to a significant number of false positives, greatly increasing the burden in the subsequent sorting and rasterization stages. This overestimation is especially pronounced for elongated Gaussians.

We identify three primary sources of false-positive intersections: 1) The semi-major and semi-minor axes of each projected Gaussian are heuristically defined as $3\sqrt{\lambda_1}$ and $3\sqrt{\lambda_2}$, where λ_1 and λ_2 ($\lambda_1 > \lambda_2$) are the eigenvalues of the 2D covariance matrix. However, in the rasterization stage, only Gaussians with opacity above the threshold contribute to rendering, reducing the effective coverage; 2) The standard approximation uses a circle with radius equal to the semi-major axis to bound the projected Gaussian, which significantly overestimates coverage along the minor axis. The use of a circumscribed square around this circle further inflates the bounding area, leading to excessive tile assignments; 3) Even within the tight bounding box of the ellipse, the geometric mismatch between the elliptical shape of the Gaussian and the rectangular shape of the box results in additional false positives.

To determine the intersection between Gaussians and tiles accurately, we propose a coarse-to-fine two-stage intersection test. In the first stage, our objective is to approximate a tight bounding box for each Gaussian at minimal computational cost, thereby filtering out the majority of non-overlapping Gaussian-tile pairs. We begin by modeling the opacity falloff of each Gaussian as a function of distance d from its projected center:

$$\alpha_i = o_i e^{-d^2/2\lambda}. \tag{3}$$

Based on this formulation, we define the effective semi-major and semi-minor axes lengths of the projected Gaussian as the distances at which its opacity decays to the threshold $\tau = \frac{1}{255}$:

$$R_{major} = \sqrt{2\ln\left(\frac{o_i}{\tau}\right)\lambda_1}, \quad R_{minor} = \sqrt{2\ln\left(\frac{o_i}{\tau}\right)\lambda_2}. \tag{4}$$

Similar to the analysis in [34], [35], the boundaries of a tight bounding box for an ellipse can be determined by locating the extrema points of the ellipse equation $F(x, y)$ along the Cartesian axes, where the partial derivatives with respect to x and y are zero:

$$\frac{\partial F(x,y)}{\partial x} = 0, \quad \frac{\partial F(x,y)}{\partial y} = 0. \tag{5}$$

By substituting these extrema points into the elliptic equation, the width W and height H of the tight bounding box are obtained through semi-major axis length R_{major} and components of the projection covariance matrix Σ'_X, Σ'_Y in the x and y directions:

$$W = 2\sqrt{\frac{\Sigma'_X}{\lambda_1}}R_{major}, \quad H = 2\sqrt{\frac{\Sigma'_Y}{\lambda_2}}R_{major}. \tag{6}$$

In the second stage, we further discard the tiles that do not actually intersect with the Gaussian within the tight bounding box. To strike a balance between accuracy and efficiency, and to avoid making the preprocessing stage a new bottleneck with complex analytical geometry, we consider l, the line connecting the center of each tile to the center of the ellipse onto the short axis of the ellipse, as shown in Fig. 8. We classify a tile as non-intersecting with the ellipse when:

$$|l|\cos\theta + r > R_{minor}, \tag{7}$$

where θ represents the angle between l and semi-minor axis of the ellipse, r represents the circumcircle radius of tiles. By simply computing and comparing the distance once, we are able to eliminate almost all false intersection pairs with minimal computational cost. As illustrated in Fig. 9, compared to prior intersection tests, our method retains substantially fewer Gaussian-tile pairs than rough screening approaches, such as GSCore's OOBB bounding box [30]. At the same time, it introduces only a negligible amount of redundancy when compared to fully accurate intersection tests like FlashGS [28], while significantly reducing the computational cost of complex geometric operations. As a result, our approach achieves optimal speedup across a variety of scenes.

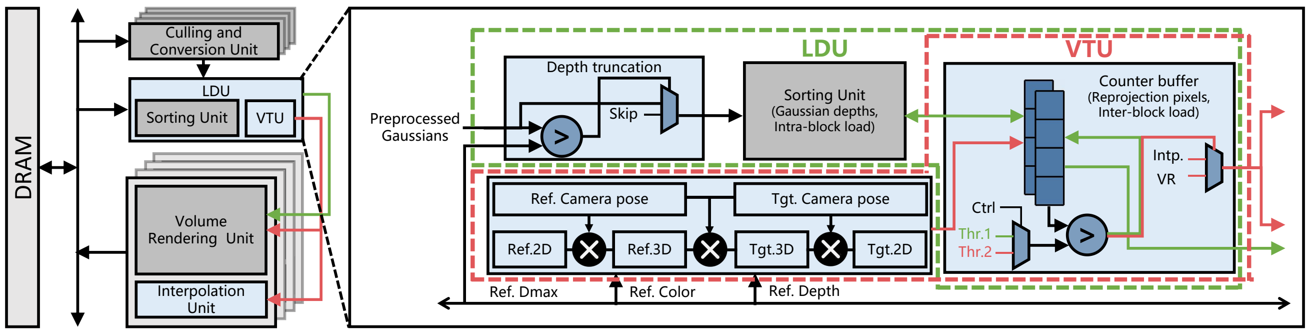


Fig. 10: Overall design of LS-Gaussian architecture. Components inherited from GSCore [30] are shown in gray, while newly introduced units are highlighted in blue. The augmented Viewpoint Transformation Unit (VTU) and Load Distribution Unit (LDU) are enclosed in red and green boxes, respectively, with colored data streams demonstrating the efficient hardware reuse capability of LS-Gaussian.

V. Streaming Hardware Architecture

Though it is undeniable that existing GPU platforms provide strong support for the parallel rendering, we identify the partial mismatch between GPU architectures and the 3DGS pipeline. Every stage of rendering heavily relies on GPU SMs, leading to frequent global memory accesses for reading and writing intermediate data. Also, different stages exhibit distinct computational characteristics and hardware demands, constraining rendering efficiency and hardware utilization. To further accelerate 3DGS, we design a dedicated hardware accelerator, LS-Gaussian, beyond the algorithmic optimizations above. The streaming mechanism enables early stages to initiate processing for subsequent frames while later stages are still executing previous ones, thereby improving parallelism and pipeline efficiency. We present the overall design (Sec. V-A) and incorporate architectural implementation at both inter-block and intra-block (Sec. V-B) levels to minimize stalls across processing units.

A. Overall Design

Fig. 10 illustrates the overall architecture of LS-Gaussian built upon GSCore [30], which comprises dedicated hardware units for preprocessing, sorting, and rasterization to enable better inter-tile overlap. On top of the original design, we enhance the Culling and Conversion Unit (CCU) responsible for Gaussian preprocessing to support our proposed two-stage accurate intersection test. This enhancement introduces a square root and logarithmic operator, while eliminating the need for GSCore's complex dual OBB Intersection Test Units (OIUs).

In addition, we enhance the Viewpoint Transformation Unit (VTU) to support sparse rendering, and extend the Load Distribution Unit (LDU) to handle task assignment and execution scheduling for each rasterization block. These enhanced components are marked in color in Fig. 10. For each viewpoint, pixel colors and maximum depths from the reference frame are retrieved for view transformation. This process involves three matrix multiplications: mapping 2D pixels to 3D space, transforming the resulting point cloud, and reprojecting it onto the new camera plane. Given that the number of pixels is substantially smaller than the number of Gaussians, this transformation incurs negligible overhead and can be parallelized with preprocessing to fully hide its latency. A counter array tracks the number of validly projected pixels per tile. When this number exceeds $\frac{5}{6}$ of the total pixels within the tile, the tile is interpolated as described in Sec. IV. Otherwise, it is forwarded to the Volume Rendering Unit (VRU) for full re-rendering. We will discuss how we balance the workload among rasterization units for rendering these tiles.

B. Load Distribution Unit

Due to the severe imbalance in tile workloads, overall hardware utilization remains suboptimal. To address this, we leverage early stopping depth prediction to implement balanced workload distribution across rasterization blocks and adopt a light-to-heavy execution order within each block. After preprocessing, Gaussians that lie beyond its predicted early stopping depth are discarded, and the remaining ones are treated as the tile's effective workload. We first compute the ideal average workload W and assign approximately N tiles per block. Tiles are then assigned to blocks sequentially. If the cumulative number of Gaussian-tile pairs in the current block exceeds $(1 + \frac{1}{N})W$, the current tile is deferred to the next block. To further enhance memory efficiency, we employ a Morton-order (Z-order) traversal strategy, which groups spatially adjacent tiles into the same block to reduce memory access overhead. Notably, this inter-block workload distribution is executed in parallel with the sorting process to avoid introducing additional latency. Considering that the Viewpoint Transformation Unit (VTU) operates in parallel with the preprocessing stage of the original pipeline, the counter array and comparators within the VTU can be directly reused, introducing no additional hardware overhead.

After completing the workload distribution, tiles within each block are sorted based on their workload, from light to heavy. Since the sorting process typically takes less time than rasterization, this ordering ensures that each tile can quickly access the pre-sorted Gaussians during the rasterization stage. It provides sufficient time for sorting high-workload tiles, preventing long sorting times from exceeding the rasterization time of lighter workload tiles, thus avoiding rasterization bubbles and reducing overall rendering time. The overhead of sorting the tiles' workloads is minimal in comparison to the Gaussian depth sorting. Therefore, we can reuse the Gaussian Sorting Unit (GSU) for this task, similar to the inter-block workload distribution design, without the need for additional hardware units.

VI. Experimental Results

A. Experimental Setup

Benchmarks. We evaluate our rendering pipeline on the Synthetic-NeRF dataset [10], which includes eight scenes with continuous viewpoints. To further validate the generality of our approach, we also select six real world scenes, including three indoor scenes (playroom, drjohnson, room) and three outdoor scenes (train, truck, garden), from three widely adopted datasets: Tanks & Temples [38], Deep Blending [39], and Mip-NeRF 360 [40]. Since these real world datasets typically contain sparse camera trajectories, we perform interpolation to construct continuous view sequences suitable for real time rendering at 90 FPS. This setup simulates camera motion at 1.8 meters and a rotational speed of 90 degrees per second.

Baselines. We use the original 3DGS running on Jetson AGX Orin as our GPU baseline. To comprehensively evaluate the effectiveness of our proposed LS-Gaussian in both algorithmic optimization and architectural implementation, we compare it with five recent 3DGS acceleration methods. Specifically, we evaluate rendering quality against Potamoi [32], another method based on viewpoint transformation. For performance evaluation, we compare GPU acceleration with AdR-Gaussian [34] and SeeLe [29], and compare our accelerator design with GSCore [30] and MetaSapiens [20].

Hardware Implementation. We implement the RTL design of LS-Gaussian hardware components using Synopsys synthesis in 16nm FinFET technology. Based on GSCore, we replace part of the multipliers and adders in the Culling and Conversion Unit (CCU) with a square root and logarithm operator. In addition, we introduce an interpolation unit, a 16KB counter buffer and other specially designed units. Benefiting from the reuse of existing hardware modules by the Load Distribution Unit (LDU), our total design area is only 1.84 mm^2, representing just a 0.39 mm^2 increase over the scaled GSCore design in 16nm (1.45mm^2), and remaining significantly smaller than Jetson-series edge GPUs (~350 mm^2) and MetaSapiens (2.73 mm^2).

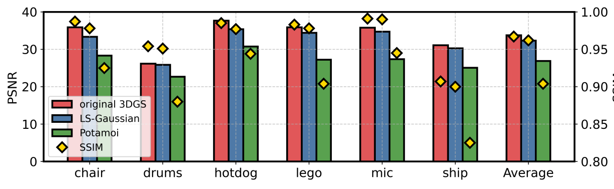
B. Rendering Quality

We compare the rendering quality of our Tile Warping-based Sparse Rendering (TWSR) against original 3DGS and Potamoi on the Synthetic-NeRF dataset, as shown in Fig. 11a. Both TWSR and Potamoi adopt a strategy of fully rendering one frame every six frames. Our method results in an average SSIM loss of only 0.005 and a PSNR loss of 1.4 dB across six scenes, which is significantly lower than Potamoi's SSIM loss of 0.063 and PSNR loss of 6.8 dB. This is because Potamoi's pixel-based inpainting ignores potentially invalid reprojections when depth priors are unavailable, resulting in incorrect floating pixels and severe visual artifacts, whereas TWSR performs tile-level full re-rendering for key regions with insufficient reprojection. In addition, due to the need for full preprocessing and sorting, Potamoi achieves limited speedups. Fig. 11b presents the visual result of a representative frame. Given that PSNR values on the Synthetic-NeRF dataset are typically above 30 dB, even more than 1.5 dB drop in TWSR does not introduce visible distortion.

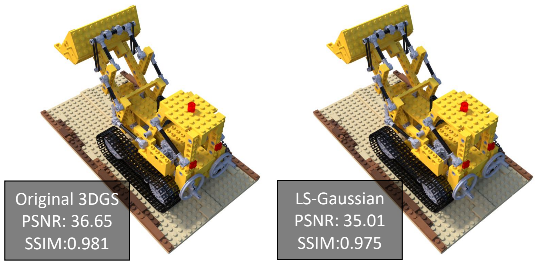
We further investigate the sensitivity of image quality and rendering speedup to the warping window size n, which defines how many frames between every two fully rendered frames, on real-world scenes. As discussed earlier, we interpolate frames in the original real-world datasets to ensure frame continuity. Therefore, we evaluate image quality by measuring the difference between rendered outputs w/ and w/o the proposed viewpoint transformation. As shown in Fig. 12a, increasing the window size leads to greater speedups but also results in image quality degradation. To balance rendering quality and efficiency, we set $n = 5$ as the default configuration for subsequent performance evaluations. The visual results and speedups of two representative frames are as shown in Fig. 12b. We observe that indoor scenes, which typically exhibit smaller depth variations, achieve both better rendering quality and higher speedups compared to outdoor scenes with more edges and corners. This phenomenon will be discussed in detail in the following subsection.

C. Performance on GPU

We evaluate the performance of LS-Gaussian deployed directly on the GPU platform using Jetson AGX Orin, as shown in Fig. 13a. Experimental results demonstrate that LS-Gaussian achieves an average

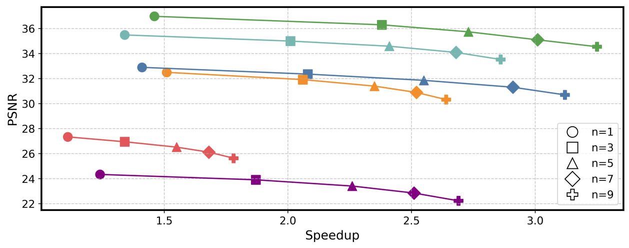


(a)

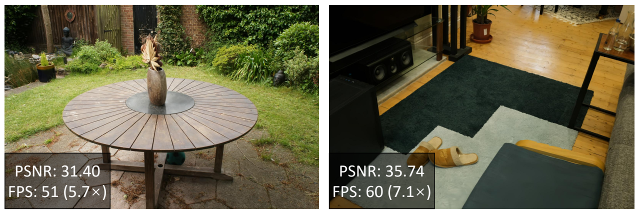


(b)

Fig. 11: (a) Comparison of the rendering quality between LS-Gaussian and original 3DGS [12]/ Potamoi [32] on the Synthetic-NeRF dataset and (b) two representative rendering output images with original 3DGS and our LS-Gaussian, respectively.



(a)

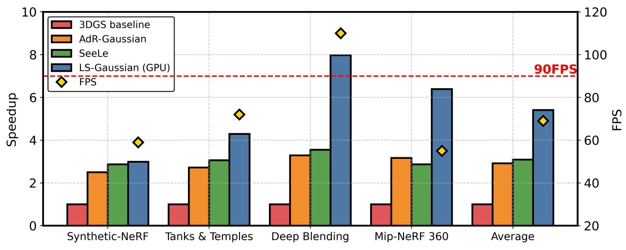


(b)

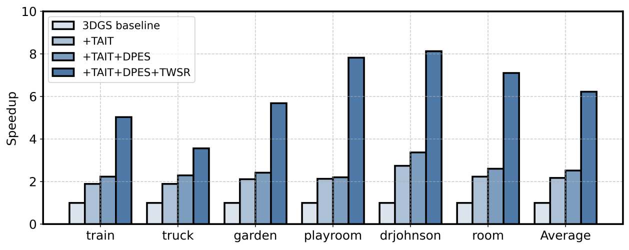
Fig. 12: (a) Speedup and PSNR (w/ and w/o TWSR) under different warping window size n on multiple real-world scenes, where each color represents a scene and (b) two representative real-world rendering images w/ TWSR, tested on Jetson AGX Orin.

speedup of 5.41× over the GPU baseline, satisfying the 90 FPS real-time rendering requirement on the Deep Blending dataset. Compared with AdR-Gaussian and SeeLe, LS-Gaussian also achieves 1.85× and 1.75× speedups, respectively. The most significant improvements are observed in indoor scenes such as playroom and drjohnson from the Deep Blending dataset. This can be attributed to the flattened structures and uniform color patterns typical of indoor environments like floors and walls, which offer higher view consistency and are more amenable to sparse rendering.

To verify this observation, we conduct an ablation study across six real-world scenes, as shown in Fig. 13b. We progressively integrate



(a)



(b)

Fig. 13: (a) Comparison of LS-Gaussian performance against prior works across multiple datasets on Jetson AGX Orin and (b) ablation study with algorithm optimizations on real-world scenes.

our algorithmic optimization techniques into the original 3DGS pipeline. The introduction of Tile Warping-based Sparse Rendering (TWSR) results in 1.56-2.35× speedups in the three outdoor scenes (train, truck, garden), while achieving 2.41-3.55× speedups in indoor scenes, confirming that the characteristics of indoor environments are more conducive to sparse rendering. The Two-stage Accurate Intersection Test (TAIT) provides an approximate 2× speedups across all scenes, highlighting its general effectiveness in reducing spurious Gaussian-tile pairs. Notably, beyond enabling accurate tile-level load estimation, Depth Prediction of Early Stopping (DPES) offers a modest speedup by saving preprocessing and sorting overhead for the next frame through depth-based culling.

D. Performance with Hardware Support

We compare our LS-Gaussian with dedicated architectural design against two recent 3DGS accelerators, GSCore and MetaSapiens, in terms of speedup over the GPU baseline. To ensure fairness, we normalize the performance of GSCore and MetaSapiens to the same 1.45 mm^2 area as LS-Gaussian using the Speedup-Area Curve reported by MetaSapiens. Since MetaSapiens does not report specific speedups in each scene, we conduct our experiments on scenes from Synthetic-NeRF, Tanks & Temples and Deep Blending, which are the same as GSCore and MetaSapiens, only reporting the average speedup of MetaSapiens. As shown in Fig. 14, LS-Gaussian achieves an average speedup of 17.3×, outperforming GSCore at 9.1× and MetaSapiens at 14.5×. We ablate the contribution of our base architecture and load distribution strategy from inter- to intra-block level, as illustrated in Fig. 15a, this improvement stems from our specialized hardware design for each processing stage and the balanced load distribution. Besides, we analyze how the hardware reuse strategy in the Load Distribution Unit (LDU) helps reduce the overall accelerator area. As shown in Fig. 15b, we report the area overhead of the augmented modules relative to GSCore. By reusing the counter buffer and comparators from the Viewpoint Transformation Unit (VTU), we achieve a 32% reduction in additional area cost. Further reuse

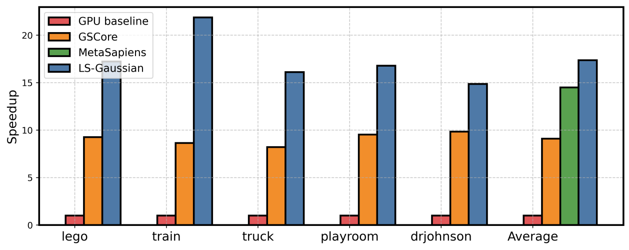
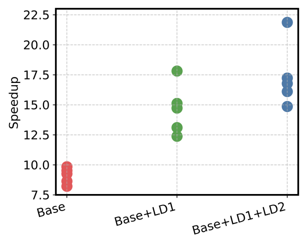
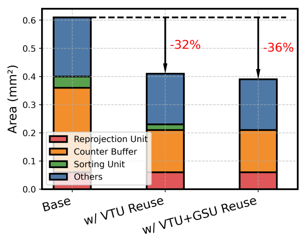


Fig. 14: Speedup comparison between our hardware and prior architectures.



(a)



(b)

Fig. 15: (a) Ablation study on the accelerator speedup with inter-block load (LD1) and intra-block load distribution (LD2), and (b) area savings of the augmented units when adopting hardware reuse.

TABLE I: Rasterization core utilization(%) comparison between original architecture and our LS-Gaussian on Synthetic-NeRF(Synthetic), Tanks & Temples (T&T), DeepBlending (DB), and Mip-NeRF 360 (Mip) datasets.

Method	Synthetic	T&T	DB	Mip	Average
Original	45.6	43.1	49.5	67.9	51.5
LS-Gaussian	88.5	89.8	78.0	98.2	88.6

of Gaussian Sorting Unit (GSU) increases the total savings to 36%, resulting in only 0.39 mm^2 of extra area.

As summarized in Tab. I, we further report the average utilization of hardware units across datasets. In contrast to sparse rendering, which favors indoor scenes, balanced load distribution yields greater performance gains in outdoor datasets such as Tanks & Temples. Unlike indoor environments, outdoor scenes exhibit significant variation in detail across regions, resulting in large workload gaps between high-frequency objects and low-frequency backgrounds, which necessitates carefully designed workload scheduling.

VII. Conclusion

In this paper, we present LS-Gaussian, a lightweight and streaming framework designed to accelerate 3D Gaussian Splatting on resource-constrained platforms. Through a systematic analysis of bottlenecks across the entire 3DGS pipeline, LS-Gaussian introduces a suite of algorithmic optimizations and architectural designs to eliminate redundant computation and mitigate hardware stalls. Extensive evaluations on both synthetic and real-world datasets demonstrate that LS-Gaussian delivers an average speedup of 5.41× over the edge GPU baseline, and up to 17.3× acceleration with dedicated hardware support, while maintaining high rendering quality.

References

[1] S. Katragadda, W. Lee, Y. Peng, P. Geneva, C. Chen, C. Guo, M. Li, and G. Huang, "NeRF-VINS: A real-time neural radiance field map-based visual-inertial navigation system," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 10 230–10 237.

[2] Y. Jiang, C. Yu, T. Xie, X. Li, Y. Feng, H. Wang, M. Li, H. Y. K. Lau, F. Gao, Y. Yang, and C. Jiang, "VR-GS: A physical dynamics-aware interactive Gaussian splatting system in virtual reality," in *Proc. of the ACM SIGGRAPH Conference and Exhibition On Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2024, p. 78.

[3] J. Cao, Z. Li, N. Wang, and C. Ma, "Lightning NeRF: Efficient hybrid scene representation for autonomous driving," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 16 803–16 809.

[4] A. Tonderski, C. Lindström, G. Hess, W. Ljungbergh, L. Svensson, and C. Petersson, "NeuRAD: Neural rendering for autonomous driving," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 14 895–14 904.

[5] Z. Qi, S. Yuan, F. Liu, H. Cao, T. Deng, J. Yang, and L. Xie, "AIR-Embodied: An efficient active 3DGS-based interaction and reconstruction framework with embodied large language model," *arXiv preprint arXiv:2409.16019*, 2024.

[6] S. Huang, L. Chen, P. Zhou, S. Chen, Z. Jiang, Y. Hu, P. Gao, H. Li, M. Yao, and G. Ren, "EnerVerse: Envisioning embodied future space for robotics manipulation," *arXiv preprint arXiv:2501.01895*, 2025.

[7] T. Chen, O. Shorinwa, W. Zeng, J. Bruno, P. M. Dames, and M. Schwager, "Splat-Nav: Safe real-time robot navigation in Gaussian splatting maps," *IEEE Transactions on Robotics*, 2025.

[8] G. Zhao, C. Ni, X. Wang, Z. Zhu, X. Zhang, Y. Wang, G. Huang, X. Chen, B. Wang, Y. Zhang, W. Mei, and X. Wang, "DriveDreamer4D: World models are effective data machines for 4d driving scene representation," *arXiv preprint arXiv:2410.13571*, 2024.

[9] C. Ni, G. Zhao, X. Wang, Z. Zhu, W. Qin, G. Huang, C. Liu, Y. Chen, Y. Wang, X. Zhang, Y. Zhan, K. Zhan, P. Jia, X. Lang, X. Wang, and W. Mei, "ReconDreamer: Crafting world models for driving scene reconstruction via online restoration," *arXiv preprint arXiv:2411.19548*, 2024.

[10] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. of European Conference on Computer Vision (ECCV)*, 2020.

[11] C. Li, S. Li, Y. Zhao, W. Zhu, and Y. Lin, "RT-NeRF: Real-time on-device neural radiance fields towards immersive AR/VR rendering," in *Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 132:1–132:9.

[12] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics (TOG)*, vol. 42, no. 4, pp. 139:1–139:14, 2023.

[13] G. Chen and W. Wang, "A survey on 3D Gaussian splatting," *arXiv preprint arXiv:2401.03890*, 2024.

[14] T. Wu, Y. Yuan, L. Zhang, J. Yang, Y. Cao, L. Yan, and L. Gao, "Recent advances in 3D Gaussian splatting," *Computational Visual Media*, vol. 10, no. 4, pp. 613–642, 2024.

[15] S. Li, C. Li, W. Zhu, B. T. Yu, Y. K. Zhao, C. Wan, H. You, H. Shi, and Y. C. Lin, "Instant-3D: Instant neural radiance field training towards on-device AR/VR 3D reconstruction," in *Proc. of Annual International Symposium on Computer Architecture (ISCA)*, 2023, pp. 6:1–6:13.

[16] Y. Feng, Z. Liu, J. Leng, M. Guo, and Y. Zhu, "Cicero: Addressing algorithmic and architectural bottlenecks in neural rendering by radiance warping and memory optimizations," in *Proc. of Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1293–1308.

[17] S. Li, Y. Zhao, C. Li, B. Guo, J. Zhang, W. Zhu, Z. Ye, C. Wan, and Y. C. Lin, "Fusion-3D: Integrated acceleration for instant 3D reconstruction and real-time rendering," in *Proc. of International Symposium on Microarchitecture (MICRO)*, 2024, pp. 78–91.

[18] Y. Liu, C. Luo, L. Fan, N. Wang, J. Peng, and Z. Zhang, "CityGaussian: Real-time high-quality large-scale scene rendering with Gaussians," in *Proc. of European Conference on Computer Vision (ECCV)*, 2024.

[19] J. Fan, W. Li, Y. Han, and Y. Tang, "Momentum-GS: Momentum Gaussian self-distillation for high-quality large scene reconstruction," *arXiv preprint arXiv:2412.04887*, 2024.

[20] W. Lin, Y. Feng, and Y. Zhu, "MetaSapiens: Real-time neural rendering with efficiency-aware pruning and accelerated foveated rendering," in *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.

[21] K. L. Navaneet, K. P. Meibodi, S. A. Koohpayegani, and H. Pirsiavash, "CompGS: Smaller and faster Gaussian splatting with vector quantization," in *Proc. of European Conference on Computer Vision (ECCV)*, 2024.

[22] F. Di Sario, R. Renzulli, M. Grangetto, A. Sugimoto, and E. Tartaglione, "GoDe: Gaussians on demand for progressive level of detail and scalable compression," *arXiv preprint arXiv:2501.13558*, 2025.

[23] G. Fang and B. Wang, "Mini-Splatting: Representing scenes with a constrained number of Gaussians," in *Proc. of European Conference on Computer Vision (ECCV)*, 2024.

[24] Z. Ye, C. Wan, C. Li, J. Hong, S. Li, L. Li, Y. Zhang, and Y. C. Lin, "3D Gaussian rendering can be sparser: Efficient rendering via learned fragment pruning," in *Proc. of Neural Information Processing Systems (NeurIPS)*, 2024.

[25] Y. Zhang, W. Jia, W. Niu, and M. Yin, "GaussianSpa: An "optimizing-sparsifying" simplification framework for compact and high-quality 3D Gaussian splatting," *arXiv preprint arXiv:2411.06019*, 2024.

[26] T. Lu, M. Yu, L. Xu, Y. Xiangli, L. Wang, D. Lin, and B. Dai, "Scaffold-GS: Structured 3D Gaussians for view-adaptive rendering," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 20 654–20 664.

[27] K. Ren, L. Jiang, T. Lu, M. Yu, L. Xu, Z. Ni, and B. Dai, "Octree-GS: Towards consistent real-time rendering with LOD-structured 3D Gaussians," *arXiv preprint arXiv:2403.17898*, 2024.

[28] G. Feng, S. Chen, R. Fu, Z. Liao, Y. Wang, T. Liu, Z. Pei, H. Li, X. Zhang, and B. Dai, "FlashGS: Efficient 3D Gaussian splatting for large-scale and high-resolution rendering," *arXiv preprint arXiv:2408.07967*, 2024.

[29] X. Huang, H. Zhu, Z. Liu, W. Lin, X. Liu, Z. He, J. Leng, M. Guo, and Y. Feng, "SeeLe: A unified acceleration framework for real-time Gaussian splatting," *arXiv preprint arXiv:2503.05168*, 2025.

[30] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "GSCore: Efficient radiance field rendering via architectural support for 3D Gaussian splatting," in *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[31] J. L. Schonberger and J.-M. Frahm, "Structure-from-Motion revisited," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] Y. Feng, W. Lin, Z. Liu, J. Leng, M. Guo, H. Zhao, X. Hou, J. Zhao, and Y. Zhu, "Potamoi: Accelerating neural rendering via a unified streaming architecture," *ACM Transactions on Architecture and Code Optimization*, vol. 21, no. 4, pp. 80:1–80:25, 2024.

[33] M. Tao, Y. Zhou, H. Xu, Z. He, Z. Yang, Y. Zhang, Z. Su, L. Xu, Z. Ma, R. Fu *et al.*, "GS-Cache: A gs-cache inference framework for large-scale Gaussian splatting models," *arXiv preprint arXiv:2502.14938*, 2025.

[34] X. Wang, R. Yi, and L. Ma, "AdR-Gaussian: Accelerating Gaussian splatting with adaptive radius," in *Proc. of the ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia (SIGGRAPH Asia)*, 2024.

[35] A. Hanson, A. Tu, G. Lin, V. Singla, M. Zwicker, and T. Goldstein, "Speedy-Splat: Fast 3D Gaussian splatting with sparse pixels and sparse primitives," *arXiv preprint arXiv:2412.00578*, 2024.

[36] S. Li, B. Keller, Y. C. Lin, and B. Khailany, "GauRast: Enhancing GPU triangle rasterizers to accelerate 3D Gaussian splatting," *arXiv preprint arXiv:2503.16681*, 2025.

[37] Z. Ye, Y. Fu, J. Zhang, L. Li, Y. Zhang, S. Li, C. Wan, C. Wan, C. Li, S. Prathipati *et al.*, "Gaussian blending unit: An edge GPU plug-in for real-time Gaussian-based rendering in AR/VR," *arXiv preprint arXiv:2503.23625*, 2025.

[38] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–13, 2017.

[39] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–15, 2018.

[40] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-NeRF 360: Unbounded anti-aliased neural radiance fields," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.